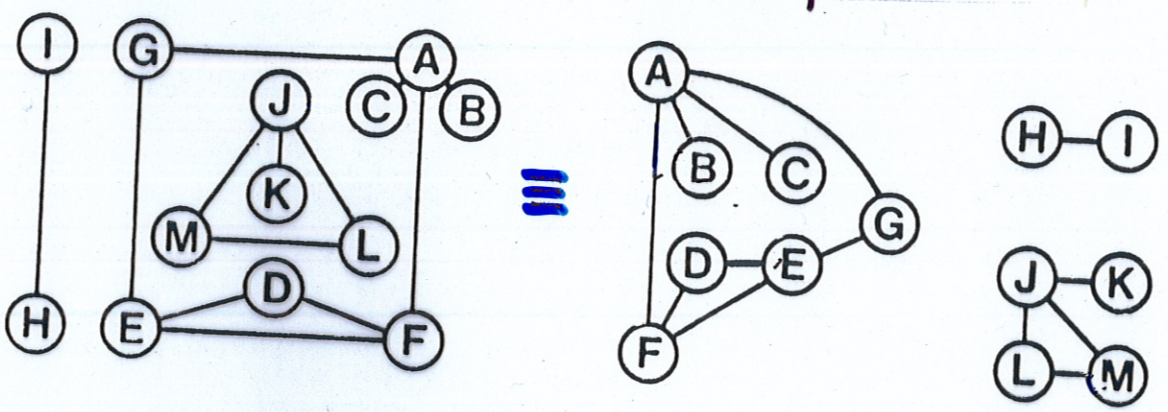
Graph representation

$G = (V, E)$

Vertex, Edge

- directed, undirected.
- weighted.
- tree $\Leftrightarrow$ connected + No cycle $\Leftrightarrow$ $\exists$1 path $u \rightsquigarrow v$



Adjacency Matrix:

	A	B	C	D	E	F	G	H	I	J	K	L	M
A	1	1	1	0	0	1	1	0	0	0	0	0	0
B	1	1	0	0	0	0	0	0	0	0	0	0	0
C	1	0	1	0	0	0	0	0	0	0	0	0	0
D	0	0	0	1	1	1	0	0	0	0	0	0	0
E	0	0	0	1	1	1	1	0	0	0	0	0	0
F	1	0	0	1	1	1	0	0	0	0	0	0	0
G	1	0	0	0	1	0	1	0	0	0	0	0	0
H	0	0	0	0	0	0	0	1	1	0	0	0	0
I	0	0	0	0	0	0	0	1	1	0	0	0	0
J	0	0	0	0	0	0	0	0	0	1	1	1	1
K	0	0	0	0	0	0	0	0	0	1	1	0	0
L	0	0	0	0	0	0	0	0	0	1	0	1	1
M	0	0	0	0	0	0	0	0	0	1	0	1	1

Adjacency List:

A: F C B G
B: A
C: A
D: F E
E: G F D
F: A E D
G: E A
H: I
I: H
J: K L M
K: J
L: J M
M: J L

Problems

- graphs
  - depth-first search, breadth-first search, priority FS
  - Cycle, connected components
  - bi-connected components
  - union-find

- weighted graphs
  - minimum spanning trees
  - Shortest paths (single source)

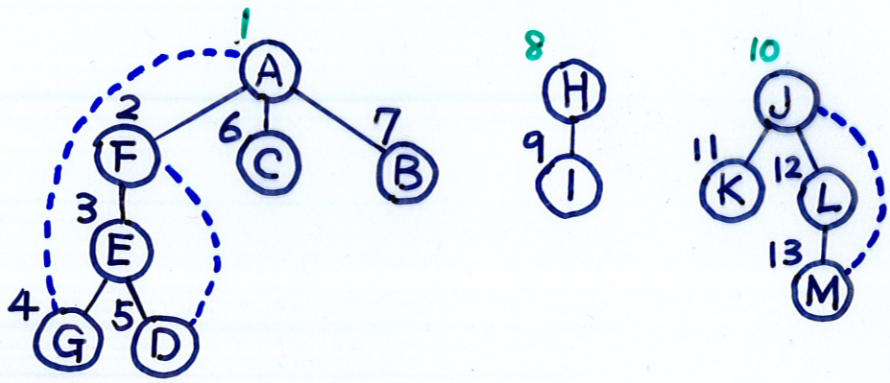
- directed graphs
  - transitive closures (connected components
    all pairs shortest paths)
  - topological sorting
  - Strongly connected components

- weighted directed graphs
  - Network flows
  - Bipartite matching
  - Stable marriage

Graph Traversal

- **Depth-First search**

```
id := 0;
for k = 1 to V do val[k] := 0;
for k = 1 to V do
    if val[k] = 0 then visit(k);

Visit(k)                          (* visits k + its descents *)
    Val[k] := ++id;
    for each son t of k do
        if val[t] = 0 then visit(t);
```



$t \neq father[k]$

- Has Cycle $\Leftrightarrow \exists$ val[t]≠0 in visit ($\exists$; line) (; to ancestor only)
- Connected Component $\Leftrightarrow$ Non-recursive visit.

- **Non-recursive DFS (stack)**

```
Visit(k)
    push(k);
    repeat
        k := pop;
        Val[k] := ++id;
        for each son t of k do
            if val[t] = 0 then
                push(t);
                val[t] := -1;
    until stack = ∅
```

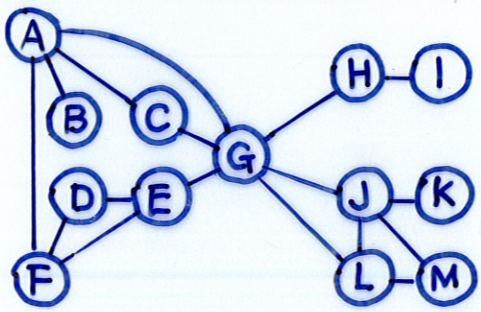
Breadth-First Search (Queue)

```
Visit(k)
    put(k);
    :
    k := get;
    .
    .
    .
    put(t);
    .
```

Priority-First Search (Graph Traversal)

- vertex **tree** – visited
  fringe – adjacent to tree vertices, but not visited yet
  unseen – not encountered (priority = unseen = ∞)

- Algorithm:
  - remove vertex x from fringe (priority queue)
  - put unseen vertices adjacent to x on fringe



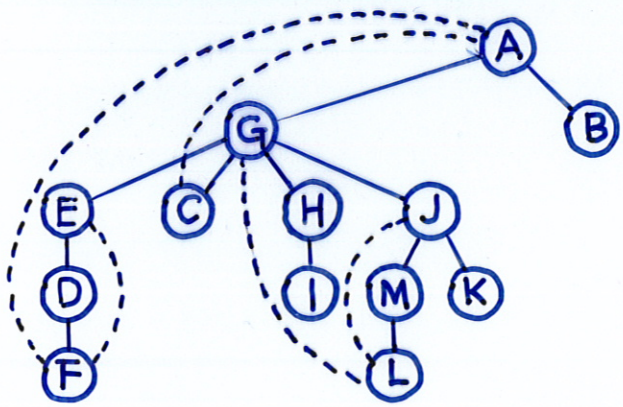
• Depth-First search

A	A						
G	G	B	C	F			
E	E	C	H	J	L	B	F
D	D	F	C	H	J	L	B
F	F	C	H	J	L	B	
C	C	H	J	L	B		
H	H	I	J	L	B		
I	I	J	L	B			
J	J	M	L	K	B		
M	M	L	K	B			
L	L	K	B				
K	K	B					
B	B						

fringe = stack

priority = V-id

high # = Low priority

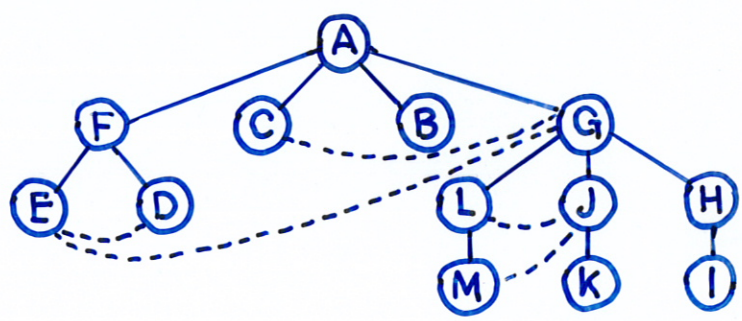


• Breadth-First search

A	A						
F	F	C	B	G	E	D	
C	C	B	G	E	D		
B	B	G	E	D	L	J	H
G	G	E	D	L	J	H	
E	E	D	L	J	H	M	
D	D	L	J	H	M	K	
L	L	J	H	M	K	I	
J	J	H	M	K	I		
H	H	M	K	I			
M	M	K	I				
K	K	I					
I	I						

Queue

priority = id



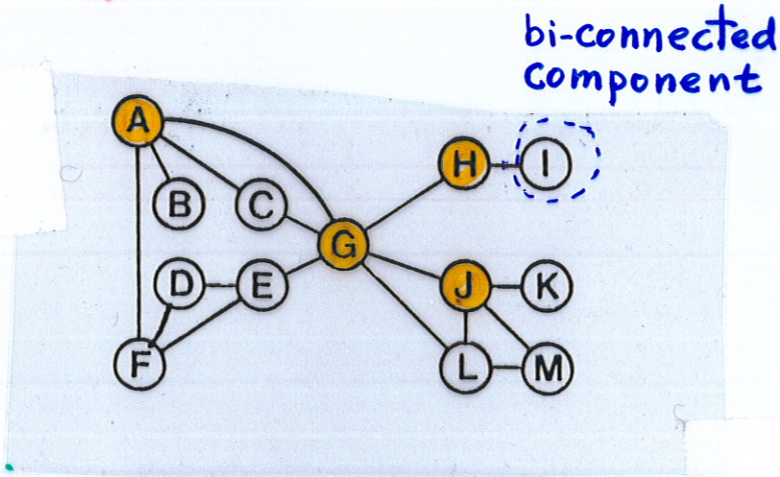
Biconnected Graph

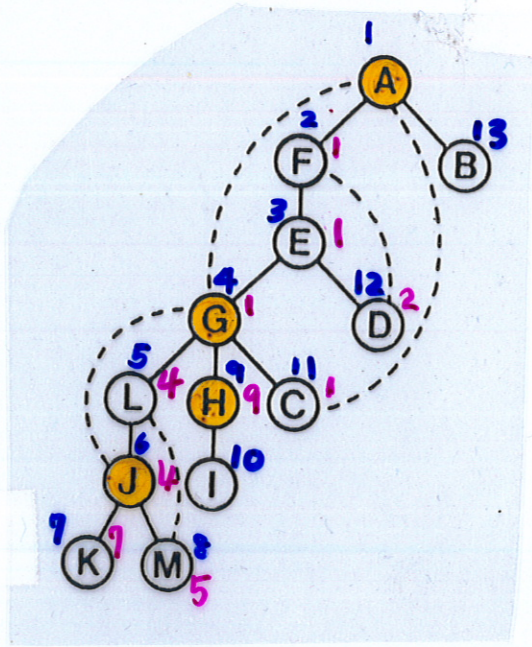
$\forall u, v$, $\exists$ 2 paths $u \overset{\frown}{\underset{\smile}{\times}} v$

$\Leftrightarrow$ No articulation point (if deleted, break G into pieces)

$\Leftrightarrow$ Delete any point, graph remains connected

• Depth-first search





- x NOT articulation point
  $\Leftrightarrow$ each son has lower node ⇢ a node higher than x

root is articulation point $\Leftrightarrow$ $\geq$ 2 sons

- Visit(K)

min = min descendant

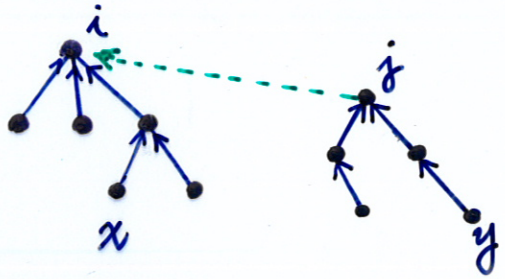
```
val[k] := ++id;   min := id;
for each son t of K do
    if val[t] = 0 then
        m := visit(t);
        if m < min then min := m;
        if m ≥ val[K] then K is articulation pt
    else
        if val[t] < min then min = val[t]

visit := min;
```

Union – Find

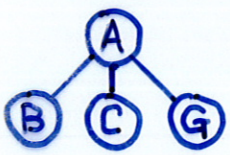
$S(x) = S(y)$?

- Find (x, y) : Are x, y in the same set? (Test if $i = j$)
- Union (x, y) := $S(x) \cup S(y)$

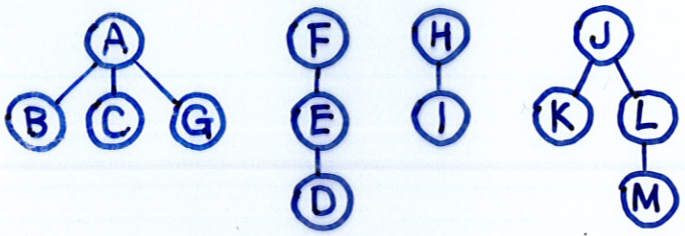
$S(x)$:= set $\ni x \Leftrightarrow$ Tree



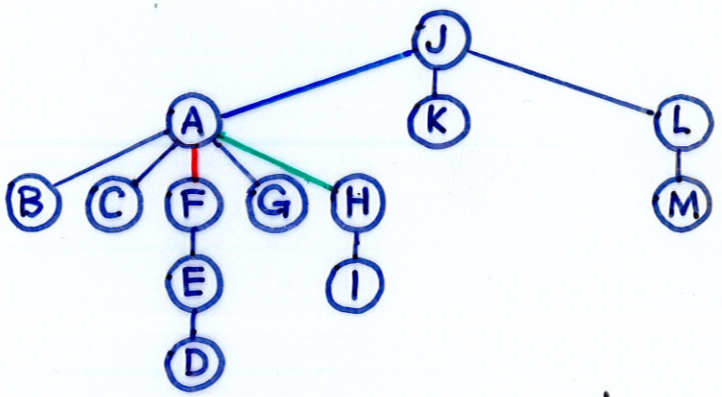
AG, AB, AC



LM, JM, JL, JK, ED, FD, HI



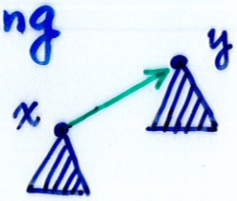
AF, FH, MG



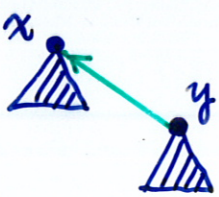
• Improvements

Weight balancing

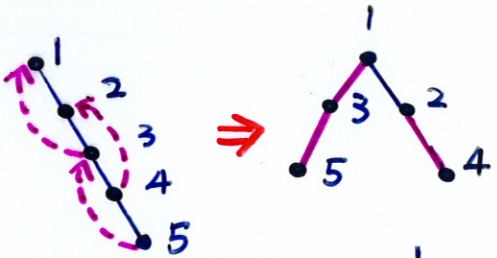
$|x| < |y|$



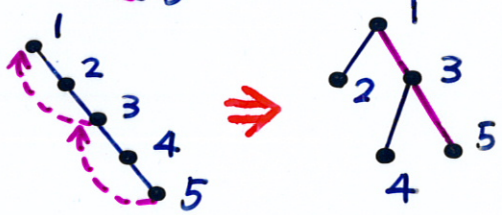
$|x| > |y|$



• Splitting



• Halving

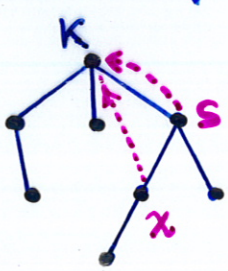


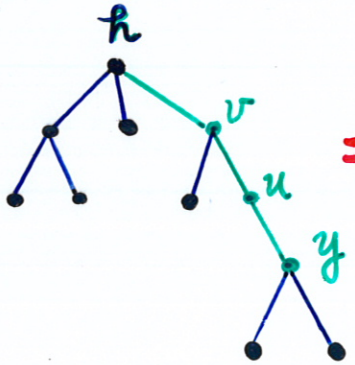
1 pass

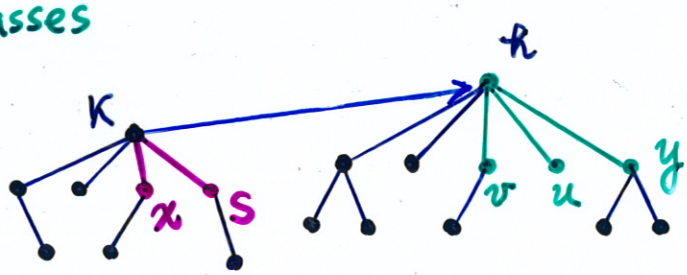
---

2 passes

Path Compression







Weighted Graph

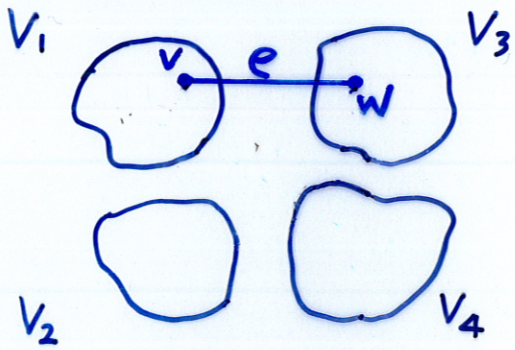
- Find **Minimum Spanning Tree** (for connected undirected graph)

  Tree: connected graph without cycle

  $\Leftrightarrow \forall u,v,\ \exists 1\ u \rightsquigarrow v$

  Spanning Tree: edges connect all vertices of G

- **Lemma**

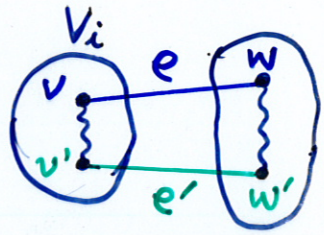


$V_1 \cup V_2 \cup V_3 \cup V_4 = V$

$e = (v, w)$ **minimum** edge, $v \in V_i$, $w \notin V_i$

$\Rightarrow \exists$ MST includes e

proof: $(T - \{e'\}) \cup e$ is spanning tree of Lower cost



- **Kruskal's** algorithm

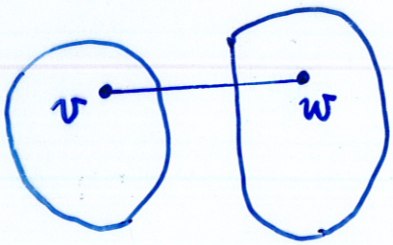
$\forall v \in V$, $\{v\}$ a component;
repeat
  add minimal edge across 2 components;
until 1 component remains

$\Leftrightarrow$

$w(e_1) \leq w(e_2) \leq \cdots \leq w(e_N)$
for $i = 1$ to N do
  if add e_i Not form cycle
    add e_i;

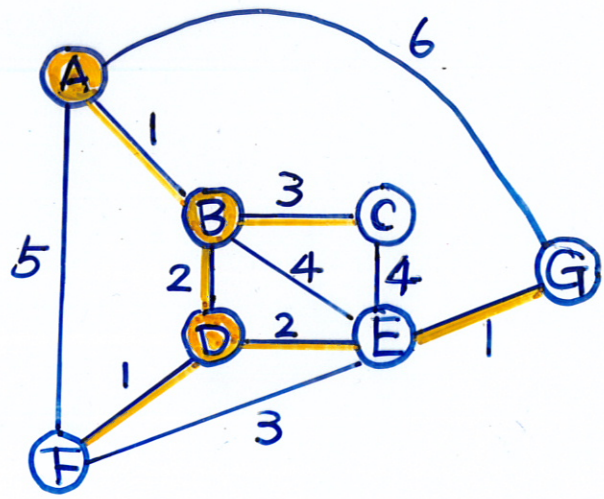
- **Prim's** algorithm:

V_1 = visited



V_2 = unvisited

$e = (v, w)$
= minimal edge visited vertex → fringe vertex



- Priority-First-Search implementation: priority = Shortest length to tree

Shortest Path Problems (directed or undirected)

(1) single pair

(2) single source : Dijkstra algorithm $O(N^2)$
: most fundamental

(3) single sink

(4) all pairs : Dijkstra algorithm $O(N^3)$
: Warshall - Floyd algo. $O(N^3)$

(2) $\Longrightarrow$ (3) : by converting edge direction

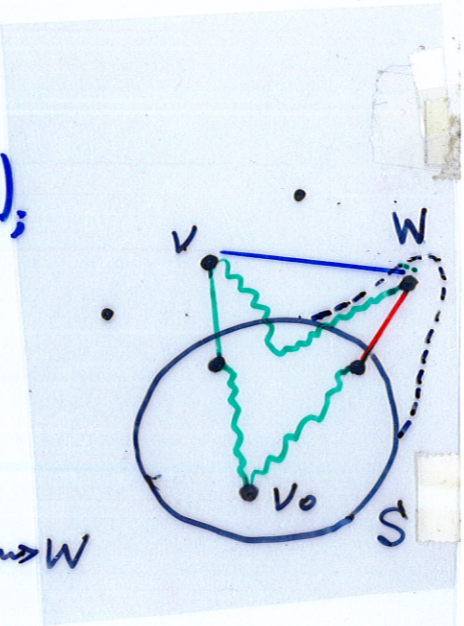
(2) $\Longrightarrow$ (1) : all known algorithms for (1) can solve (2)

(2) $\Longrightarrow$ (4) : apply N times

Shortest Path (Single Source)

- **Unweighted Graph**: Breadth-First Search
- **Weighted Graph**: Dijkstra's algorithm

```
S ← {v0}; for each v ∈ V do D[v] ← d(v0, v);
While S ≠ V do {
    choose w ∉ S, D[w] min;
    S ← S ∪ {w};
    for each v ∈ V − S do
        D[v] ← min(D[v], D[w] + d[w, v]);
}
```



Property:

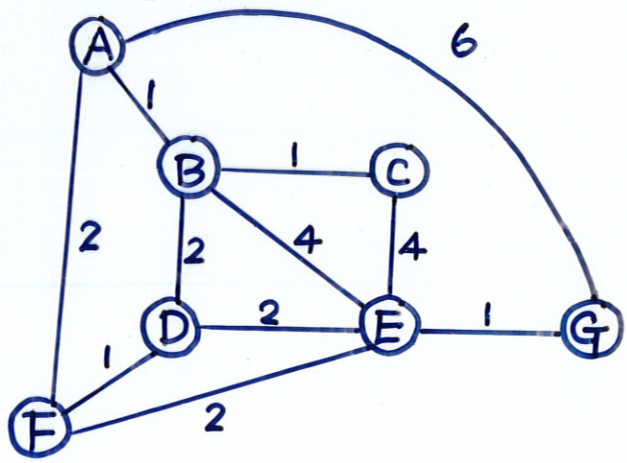
$v \in S \Rightarrow D[v]$ = shortest path $v_0 \rightsquigarrow v$

$v \notin S \Rightarrow D[v]$ = " $v_0 \rightsquigarrow x \rightarrow v$ (in S)

Proof: Otherwise, $\exists$ $v_0 \rightsquigarrow v$ (in S) $+ v \rightsquigarrow w < v_0 \rightsquigarrow w$

- **Example**

S	A	B	C	D	E	F	G
A	0	1	∞	∞	∞	2	6
A B	0	1	2	3	5	2	6
A B F	0	1	2	3	4	2	6
A B C F	0	1	2	3	4	2	6
A B C D F	0	1	2	3	4	2	6
A B C D E F	0	1	2	3	4	2	5
A B C D E F G	0	1	2	3	4	2	5



- For both directed and undirected graph
- $T = O(N^2)$
- Priority-First-Search implementation: Priority = $D[w] + d(w, v)$
  - priority queue = ϕ; put(S);
  - only modify D[v], for W → V

Directed Graph $G=(V, E)$

- **Transitive closure** (connected cmp for undirected G)
  - $A = (a_{ij})_{n\times n}$ $\quad a_{ij}=1 \Leftrightarrow i \to j \Leftrightarrow (i,j)\in E$
  - $A^2 = B = (b_{ij})$ $\quad b_{ij} = a_{i1}\cdot a_{1j} + a_{i2}\cdot a_{2j} + \cdots + a_{in}\cdot a_{nj}$ (OR, and)
    $= 1 \Leftrightarrow i \to k \to j$
  - $A^3 = B\cdot A = (c_{ij})$ $\quad c_{ij} = b_{i1}\cdot a_{1j} + b_{i2}\cdot a_{2j} + \cdots + b_{in}\cdot a_{nj}$
    $= 1 \Leftrightarrow i \to k \to l \to j$
  - $\vdots$
  - $A^* = A + A^2 + \cdots + A^{n-1} + A^n + \cdots$
    $= A + A^2 + \cdots + A^{n-1}$

  $G^* = (V, E^*)$ transitive closure of G $\quad a^*_{ij} = 1 \Leftrightarrow i \rightsquigarrow j$

- **Shortest path** (all pairs)
  - $A = (a_{ij})_{n\times n}$ $\quad a_{ij} = \text{dist}(i, j)$
  - $A^2 = (b_{ij})$ $\quad b_{ij} = (a_{i1} + a_{1j}) \min (a_{i2} + a_{2j}) \min \cdots \min (a_{in} + a_{nj})$
    $=$ shortest distance $i \to k \to j$
  - $A^3 = (c_{ij})$ $\quad //$ $\quad i \to k \to l \to j$
  - $\vdots$
  - $A^* = (a^*_{ij})$ $\quad a^*_{ij} =$ shortest distance $i \rightsquigarrow j$

- $T = O(N^4)$
- If apply Dijkstra's algorithm N times $\Rightarrow T = O(N^3)$

Warshall's algorithm (Dynamic Programming)

- **Transitive closure** $G=(V,E)$
  - $a[x,y]=1 \Leftrightarrow (x,y)\in E$

After each j, $a[x,y]=1 \Leftrightarrow x \rightsquigarrow y$ in $[1,j]$

```
for j = 1 → N do
    for x = 1 → N do
        for y = 1 → N do
            a[x,y] := a[x,y] OR a[x,j] AND a[j,y]
```

$a[x,y]$: $x \rightsquigarrow y$ in $[1,j-1]$

$x \rightsquigarrow y$ in $[1,j] \Leftarrow$ $x \rightsquigarrow y$ in $[1,j-1]$ OR $x \rightsquigarrow j$ AND $j \rightsquigarrow y$

  - $a^*[x,y]=1 \Leftrightarrow x \rightsquigarrow y$

$G^*=(V,E^*)$ transitive closure of $G=(V,E)$

- **Shortest Path** (all pairs) (Floyd)

$a[i,j] = dist(i,j)$

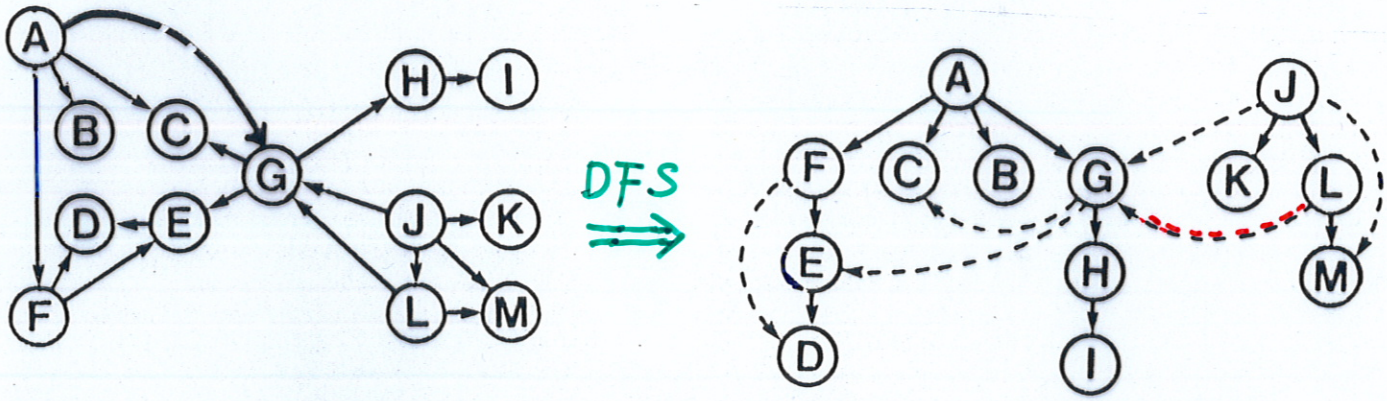
$\Downarrow$ $a[x,y] := \text{Min}(a[x,y],\ a[x,j]+a[j,y])$

$a^*[i,j]$ = shortest path distance

$T = O(VE+V^2)$

Topological Sorting (Directed acyclic graph)

- Topological order : i before j $\Leftarrow$ $i \rightarrow j$ in graph



Example: J K L M A G H I F E D B C

A J L F K G E M B H C I D

- Algorithm: produce reverse topological order by DFS

```
Visit(K)
    val[K] := ++id;
    for each son t of K do
        if val[t]=0 then visit(t)
    Print(K);
```

$\Rightarrow$ D E F C B I H G A K M L J

- Algorithm':

	count	Link
A	0	→ B, C, F, G
B	1	·
C	2	·
D	2	·
E	2	→ D
F	1	→ E, D
G	3	→ E, C, H
H	1	→ I

output A $\Rightarrow$

	count	Link
A	0	→ B, C, F, G
B	0	·
C	1	·
D	2	·
E	2	→ D
F	0	→ E, D
G	2	→ E, C, H
H	1	→ I

- No count(v) = 0 $\Rightarrow$ directed cycle exists

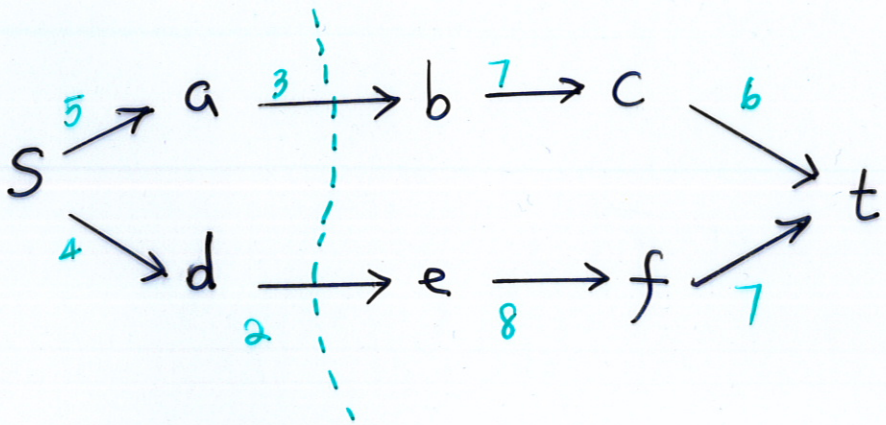
Strongly connected components

- G: Strongly connected : $\forall u, v, \exists\ u \rightsquigarrow v$

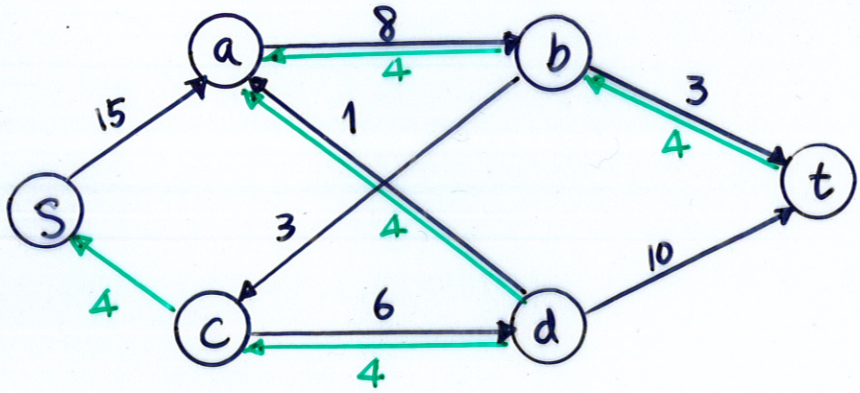
1 - dim

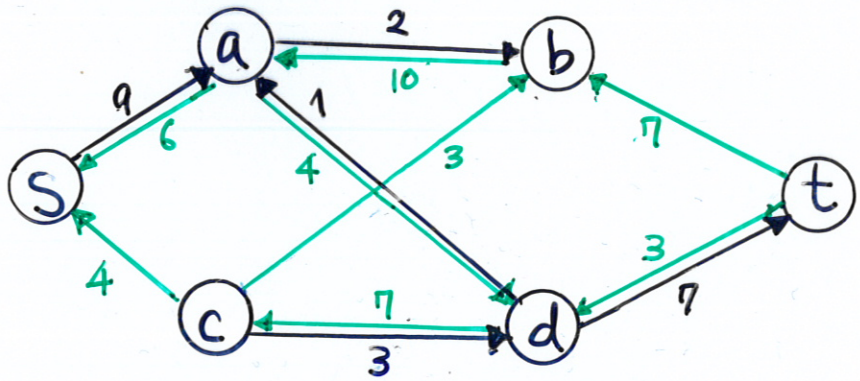
bottle neck

Ford - Fulkerson th: min cut = max flow



2 - dim





- Residual Network
- $\exists$ path $s \rightsquigarrow t$?

Network Flow

- network : directed graph with $\begin{matrix} s: \text{source} \\ t: \text{sink} \end{matrix}$ + **capacity**
- flow f : $\forall$ edge e, $\quad 0 \le f(e) \le c(e)$

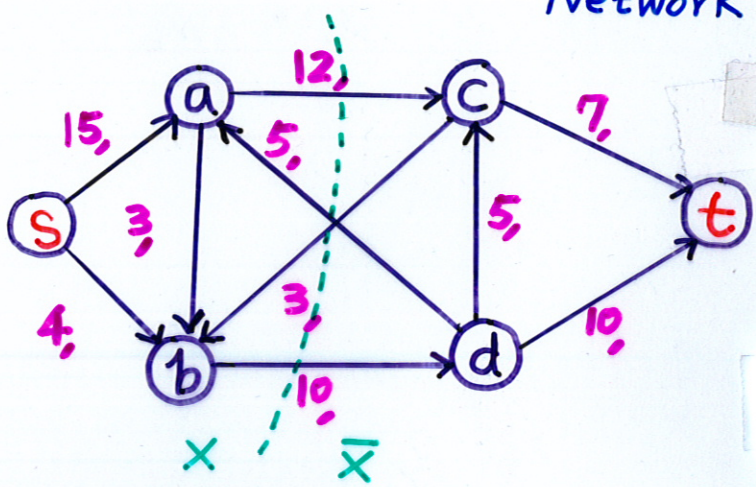
  $\forall$ vertex $v \neq s,t$, $\quad \sum_i f(i,v) = \sum_j f(v,j)$

value of $f = |f| := \sum_i f(i,t) = \sum_j f(s,j)$

- **Maximal flow** problem :

  find flow with max value

Network



- cut $(X, \bar{X})$ $\begin{cases} \bar{X} = V - X \\ s \in X,\ t \in \bar{X} \end{cases}$

$$c(X \to \bar{X}) = \sum_{v \in X,\, w \in \bar{X}} c(v,w) \quad : \text{ cut capacity}$$

$$f(X \to \bar{X}) = \sum_{v \in X,\, w \in \bar{X}} f(v,w) \quad : \text{ flow across the cut}$$

- **Lemma** : $|f| = f(X \to \bar{X}) - f(\bar{X} \to X) \le c(X \to \bar{X})$

  Pf : $|f| = |f| + 0 + 0 = \sum_{s,a,b} \text{out}(v) - \sum_{s,a,b} \text{in}(v)$

  $= f(X \to \bar{X}) - f(\bar{X} \to X)$

- **Max-flow, min-cut Theorem** (Ford, Fulkerson)

  max-flow $\Leftrightarrow |f| = c(X \to X) \Leftrightarrow$ $\begin{matrix} c,c \to \\ \leftarrow c,0 \end{matrix}$ saturated $\Leftrightarrow$ No a.p.

  $\Leftrightarrow$ min-cut. empty

- augmenting path

$s \xrightarrow[+7]{12,5} a \xrightarrow[+5]{8,3} b \xrightarrow[+5]{7,2} t \quad +5, \quad \xrightarrow{c,f} \ f(e) < c(e)$

$s \xrightarrow[+7]{12,5} a \xleftarrow[-3]{8,3} b \xrightarrow[+5]{7,2} t \quad +3, \quad \xleftarrow{c,f} \ f(e) > 0$

$\downarrow 8 \qquad \uparrow 5$

Network Flow

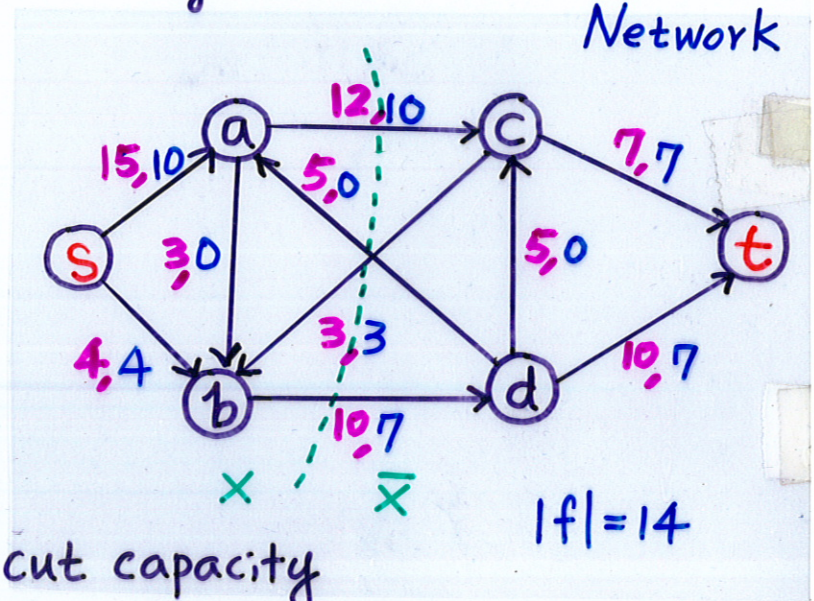
- network : directed graph with $\begin{matrix} s: \text{source} \\ t: \text{sink} \end{matrix}$ + **capacity**
- flow f : $\forall$ edge e, $\quad 0 \le f(e) \le c(e)$

  $\forall$ vertex $v \neq s,t$, $\quad \sum_i f(i,v) = \sum_j f(v,j)$

value of $f = |f| := \sum_i f(i,t) = \sum_j f(s,j)$

- **Maximal flow** problem :

  find flow with max value



- cut $(X, \bar{X})$ $\begin{cases} \bar{X} = V - X \\ s \in X,\ t \in \bar{X} \end{cases}$

$c(X \to \bar{X}) = \sum_{v \in X,\, w \in \bar{X}} c(v,w)$: cut capacity

$f(X \to \bar{X}) = \sum_{v \in X,\, w \in \bar{X}} f(v,w)$: flow across the cut

- **Lemma** : $|f| = f(X \to \bar{X}) - f(\bar{X} \to X) \quad \le c(X \to \bar{X})$

Pf : $|f| = |f| + 0 + 0 = \sum_{s,a,b} \text{out}(v) - \sum_{s,a,b} \text{in}(v)$

$= f(X \to \bar{X}) - f(\bar{X} \to X)$

- **Max-flow, min-cut Theorem** (Ford, Fulkerson)

max-flow $\Leftrightarrow |f| = c(X \to \bar{X}) \Leftrightarrow$ $\begin{matrix} c,c \to \\ \leftarrow c,0 \end{matrix}$ saturated $\Leftrightarrow$ No a.p.

$\Leftrightarrow$ min-cut. empty

- augmenting path

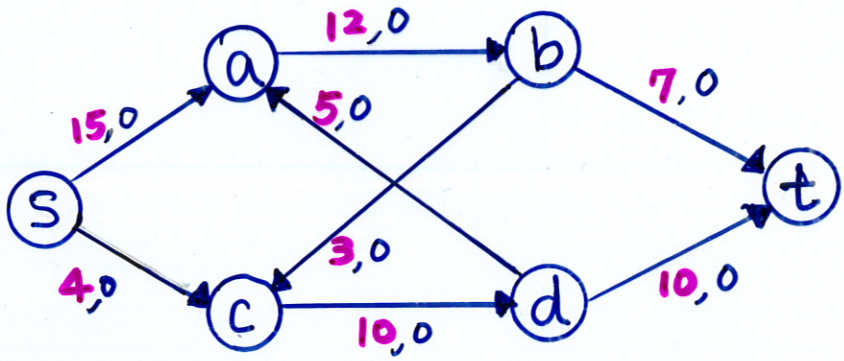
$s \xrightarrow[+7]{12,5} a \xrightarrow[+5]{8,3} b \xrightarrow[+5]{7,2} t \quad +5, \quad \xrightarrow{c,f} \ f(e) < c(e)$

$s \xrightarrow[+7]{12,5} a \xleftarrow[-3]{8,3} b \xrightarrow[+5]{7,2} t \quad +3, \quad \xleftarrow{c,f} \ f(e) > 0$

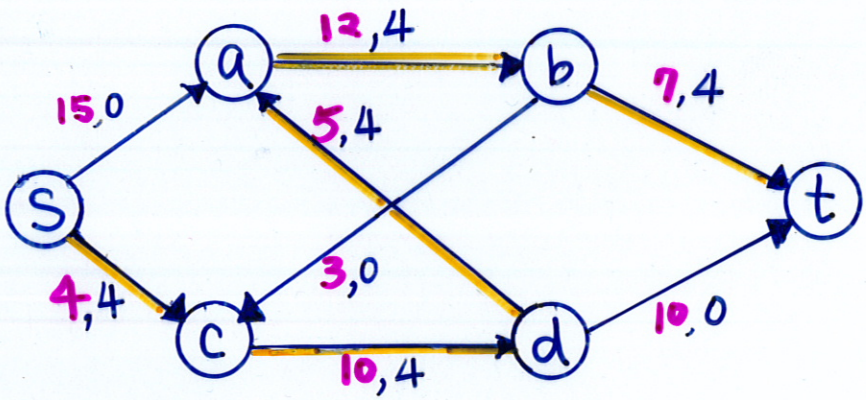
$\downarrow 8 \qquad \uparrow 5$

- **Labelling algorithm:**
  find possible augmenting path, increase flow

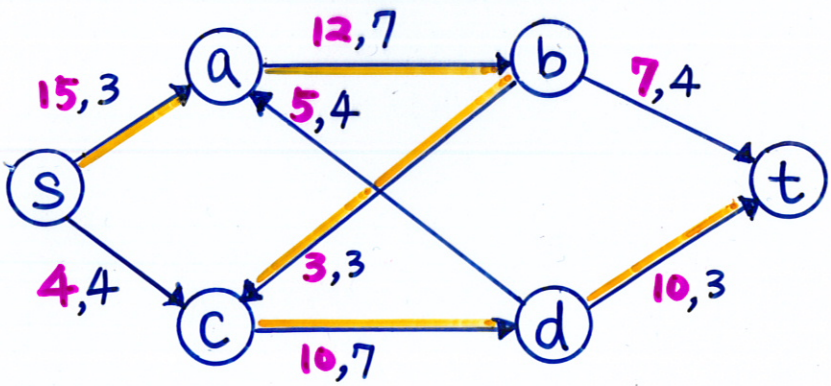
- **Example:**



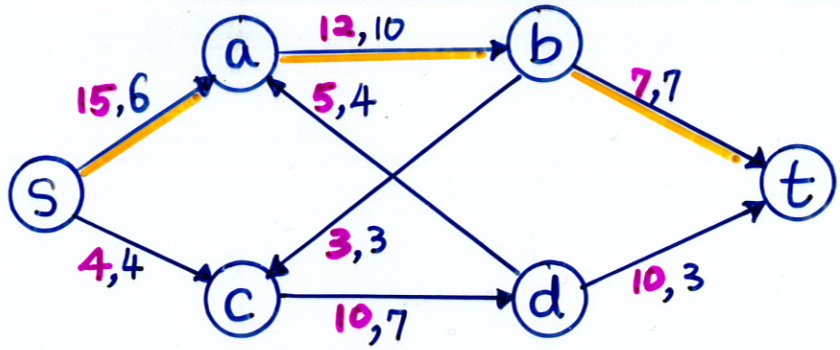
$s \xrightarrow{4} c \xrightarrow{10} d \xrightarrow{5} a \xrightarrow{12} b \xrightarrow{7} t$ +4



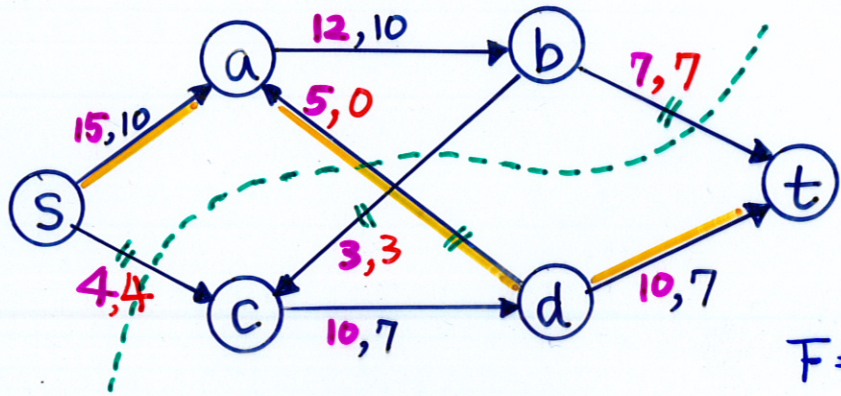
$s \xrightarrow{15} a \xrightarrow{8} b \xrightarrow{3} c \xrightarrow{6} d \xrightarrow{10} t$ +3



$s \xrightarrow{12} a \xrightarrow{5} b \xrightarrow{3} t$ +3

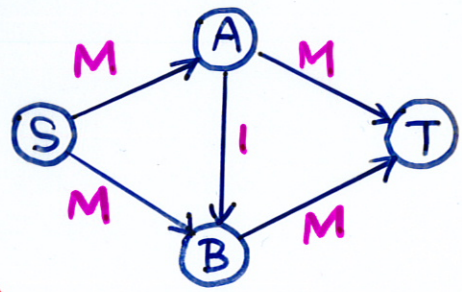


$s \xrightarrow{9} a \xleftarrow{-4} d \xrightarrow{7} t \quad +4$



$F = 14$

- **Drawbacks:**
  - Irrational Capacity: might never terminate
  - Even Integral Capacity: $2M$ increments



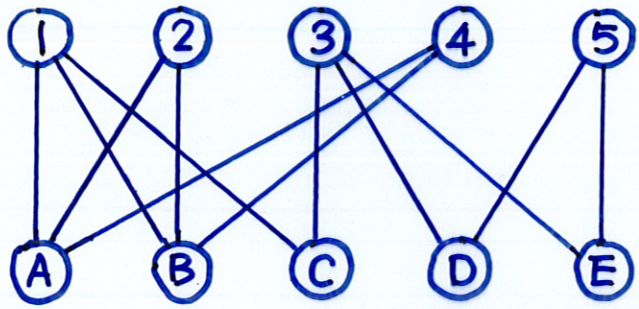
- **Edmond, Karp:**
  - Breadth-First-Search + Shortest path $T = O(V^3E)$
  - Increase the Flow by Largest amount
    - priority-first-search implementation:
      - $K \longrightarrow t$, $size[K,t] > 0 \Rightarrow$ priority $= size[K,t] - flow[K,t]$
        $K \longleftarrow t$, " $< 0 \Rightarrow$ priority $= -flow[K,t]$;
        if priority $>$ val[K] then priority $=$ val[K];
        ↑ Max increase $S \rightarrow K$
      - search stops when sink found;
    - Efficient in practice.

Matching

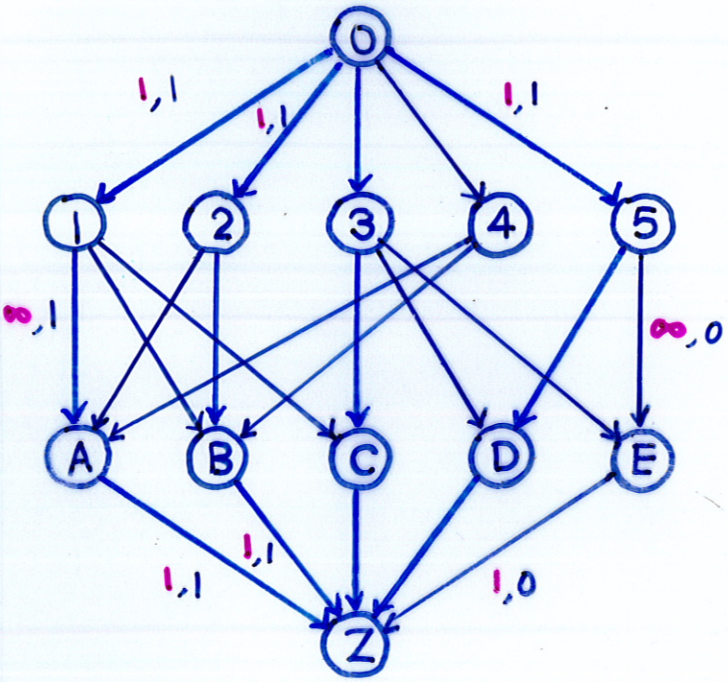
: subset of edges where no edges incident

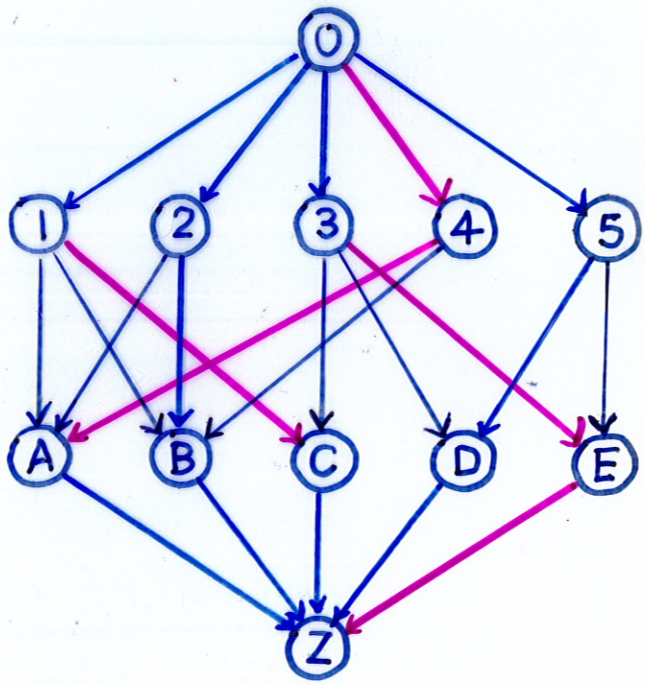
Goal: Find maximal matching (weighted / unweighted)

- Bipartite graph



- Applying Network Flows



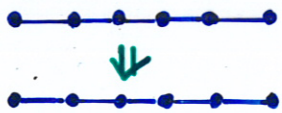


- Integral capacity has a maximal integral flow

- maximal matching = maximal flow

- augmenting path 0 4 A 1 C 3 E Z in network flow

  = augmenting path 4 A 1 C 3 E in Hungarian matching method



Stable Marriage

- Preference List

Male

A:	2	5	1	3	4
B:	1	2	3	4	5
C:	2	3	5	4	1
D:	1	3	2	4	5
E:	5	3	2	1	4

Female

1:	E	A	D	B	C
2:	D	E	B	A	C
3:	A	D	B	C	E
4:	C	B	D	A	E
5:	D	B	C	E	A

- **Matching (Marriage)**: 1-1, onto mapping
- **Unstable** marriage: $\exists$ a 1, b 2 s.t. a: 2 1; 2: a b
- Example: A1, B3, C2, D4, E5 is unstable
- **Goal**: Find a stable matching
- Algorithm

  Each man M in turn becomes suitor, proposes to each W on his list

  (1) W is free $\Rightarrow$ MW engaged

  (2) For W, M $\gg$ current fiancee M' $\Rightarrow$ MW engage, M' becomes suitor

  (3) M' $\gg$ M $\Rightarrow$ M proposes to next W on his list

- Analysis

  (1) Program terminates ?

  (2) Generate a Matching ?

  (3) Stable ?

  (4) **Male optimal** : No man does better in other stable matchig

  (5) **Female pessimal**: No woman does worst "

Dynamic Programming

: solve a problem by solving all possible sub-problems.

• Knapsack problem

$N = 5$: # items
$M = 17$: total capacity

Name	A	B	C	D	E
size	3	4	7	8	9
value	4	5	10	11	13

- 5A ⇒ value = 20
- D+E ⇒ value = 24

cost[i] = highest value for capacity i.
best[i] = last item added

• Program

```
for j = 1 to N
  for i = 1 to M
    t := cost[i - size[j]] + val[j]      ← last item is j
    if t > cost[i] then                  ← best for rest of Knapsack
      cost[i] := t;
      best[i] := j;
```

(* use first j items only *)
(* compute each capacity i *)

i →

j ↓	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	
	0	0	4	4	4	8	8	8	12	12	12	16	16	16	20	20	20	A only
			A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	
	0	0	4	5	5	8	9	10	12	13	14	16	17	18	20	21	22	A, B "
			A	B	B	A	B	B	A	B	B	A	B	B	A	B	B	
	0	0	4	5	5	8	10	10	12	14	15	16	18	20	20	22	24	A, B, C
			A	B	B	A	C	B	A	C	C	A	C	C	A	C	C	only
	0	0	4	5	5	8	10	11	12	14	15	16	18	20	21	22	24	A, B, C, D
			A	B	B	A	C	D	A	C	C	A	C	C	D	C	C	"
	0	0	4	5	5	8	10	11	13	14	15	17	18	20	21	23	24	" E
			A	B	B	A	C	D	E	C	C	E	C	C	D	E	C	

10+11 = 21 > 20

∴ Best choice : C + C + A for M = 17

• Matrix chain Product

- pqr "*" for $[a_{ij}]_{p\times q}[b_{ij}]_{q\times r} = [c_{ij}]_{p\times r}$, $c_{ij} = \sum_{k=1}^{q} a_{ik} * b_{kj}$
- $\begin{bmatrix} x \\ y \end{bmatrix}[a][b] = \begin{bmatrix} xa \\ ya \end{bmatrix}[b] = \begin{bmatrix} (xa)b \\ (ya)b \end{bmatrix}$ $\quad 2\cdot1\cdot1 + 2\cdot1\cdot1 = 4$ "*"

  $= \begin{bmatrix} x \\ y \end{bmatrix}[ab] = \begin{bmatrix} x(ab) \\ y(ab) \end{bmatrix}$ $\quad 1\cdot1\cdot1 + 2\cdot1\cdot1 = 3$

• In General

Minimize total # "*" among $C_N = \frac{1}{N}\binom{2N-2}{N-1}$ ways for $M_1 M_2 M_3 \cdots M_N$

$r_1\times r_2 \quad r_2\times r_3 \cdots r_N\times r_{N+1}$

catalan #: $C_N = C_1 C_{N-1} + C_2 C_{N-2} + \cdots + C_{N-1} C_1$

(1) 1 way for $M_1M_2,\ M_2M_3,\ \ldots,\ M_{N-1}M_N$

cost $r_1r_2r_3 \quad r_2r_3r_4 \quad \ldots \quad r_{N-1}r_Nr_{N+1}$

(2) $M_1\overline{M_2M_3},\ M_2\overline{M_3M_4},\ \ldots,\ M_{N-2}\overline{M_{N-1}M_N}$

$$\text{Min}\begin{cases} r_1r_2r_3 + r_1r_3r_4 & r_2r_3r_4 + r_2r_4r_5 & r_{N-2}r_{N-1}r_N + r_{N-2}r_Nr_{N+1} \\ r_2r_3r_4 + r_1r_2r_4 & r_3r_4r_5 + r_2r_3r_5 & r_{N-1}r_Nr_{N+1} + r_{N-2}r_{N-1}r_{N+1} \end{cases}$$

(3) $M_1M_2M_3M_4,\ M_2M_3M_4M_5, \ldots$

(4) Program

cost$[\ell, r]$:= min cost for $M_\ell \cdots M_r$

$\underbrace{M_i M_{i+1} \cdots M_{k-1}}\ \underbrace{M_k \cdots M_{i+j}}$

for $1 \le j \le N-1$
  for $1 \le i \le N-j$

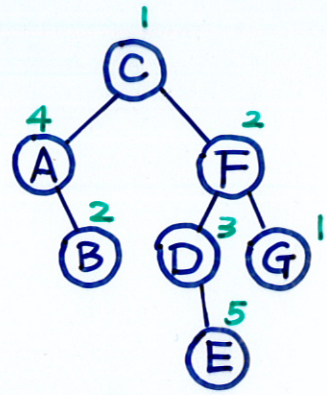
$$\text{Cost}[i, i+j] := \min_{i+1\le k\le i+j} \{\text{Cost}[i, k-1] + \text{Cost}[k, i+j] + r_i r_k r_{i+j+1}\}$$

best$[i, i+j]$:= k;

In particular:

$$\text{Cost}[i, i+1] := \underset{0}{\underset{\parallel}{\text{Cost}[i, i]}} + \underset{0}{\underset{\parallel}{\text{Cost}[i+1, i+1]}} + r_i r_{i+1} r_{i+2}$$

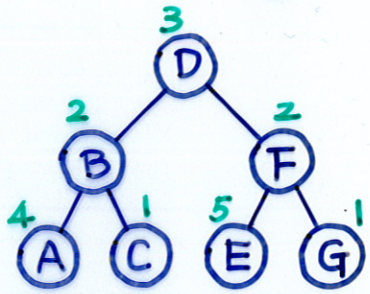
• Optimal Binary Search Tree



- Weighted Internal Path Length (WIPL)

$$WIPL = 4\cdot2 + 2\cdot3 + 1\cdot1 + 3\cdot3 + 5\cdot4 + 2\cdot2 + 1\cdot3 = 51$$

$$WIPL = 3\cdot1 + (2+2)\cdot2 + (4+1+5+1)\cdot3 = 44$$



ABC D EFG

- Goal : Given Keys: $K_1 < K_2 < \cdots < K_N$

  Frequency: $r_1 \quad r_2 \quad r_N$

  Arrange keys in Binary Search Tree to minimize WIPL

- Remark:

  (1) # binary tree with N nodes $C_N = \frac{1}{N+1}\binom{2N}{N}$

  $\left(C_N = C_{N-1} + C_1 C_{N-2} + C_2 C_{N-3} + \cdots + C_{N-2} C_1 + C_{N-1}\right)$

  (2) Huffman coding : no need to maintain key order

- Algorithm (similar to matrix *)

```
for 1 ≤ j ≤ N-1
  for 1 ≤ i ≤ N-j
```

minimize $K_i \cdots K_{k-1}, K_k, K_{k+1} \cdots K_{i+j}$ (root)

$$Cost[i, i+j] := \min_{i \le k \le i+j} \{ Cost[i, k-1] + (r_i + \cdots + r_{k-1}) + r_k + (r_{k+1} + \cdots + r_{i+j}) + Cost[k+1, i+j] \}$$

$$= \sum_{i \le k \le i+j} r_k + \min_{i \le k \le i+j} \{ Cost[i, k-1] + Cost[k+1, i+j] \}$$

```
best[i, i+j] := k;
```

Initially, $Cost[i, i] := r_i; \quad 1 \le i \le N$

$Cost[i, i-1] := 0; \quad 1 \le i \le N+1$

What is Computation ? & NP- complete problems.

- What is **computation, algorithm, program** ?

Church thesis : Computable function

Computation $Y = f(x)$

= Turing (machine) Computable function
= Kleene's partial recursive function
= Post correspondence system

- **Recursive function**

(1) Base functions : $n(x) = 0$ zero **are recursive**

$S(x) = x+1$ successor

$U_i^n(x_1, \dots, x_n) = x_i$ projection

(2) Operators: f, g_i are recursive $\Rightarrow$ h **is recursive**

composition: $h(x_1, \dots, x_n) = f(g_1(x_1, \dots, x_n), \dots, g_K(x_1, \dots, x_N))$

recursion:
$$\begin{cases} h(x_1, \dots, x_N, 0) = f(x_1, \dots, x_N) \\ h(x_1, \dots, x_N, y+1) = g(y, h(x_1, \dots, x_N, y), x_1, \dots, x_N) \end{cases}$$

minimization :

Example:

(1) $a(x, y) = x + y$ is recursive

$a(x, 0) = x = U_1^1(x)$

$a(x, y+1) = a(x, y) + 1 = S\left(U_2^3(y, a(x, y), x)\right)$

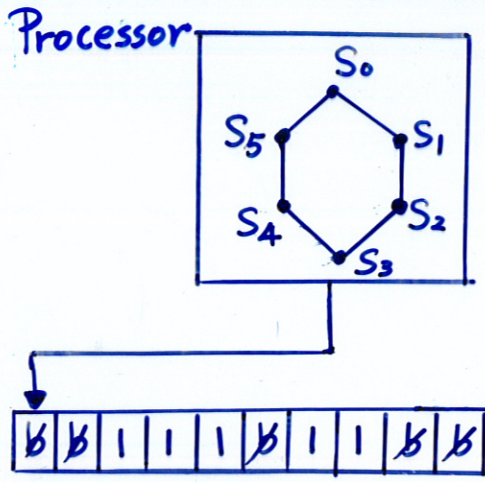
(2) $p(x, y) = x \cdot y$ is recursive

$p(x, 0) = 0$

$p(x, y+1) = p(x, y) + x = a\left(U_2^3(y, p(x, y), x), U_3^3(y, p(x, y), x)\right)$

- Partial recursive functions $\Leftrightarrow$ Turing computable function

- **Turing Machine**



- States : $S = \{ S_0, S_1, S_2, S_3, S_4, S_5 \}$
- Tape Symbols : $T = \{ b, 1 \}$
- Transitions :

compute : $x+y$

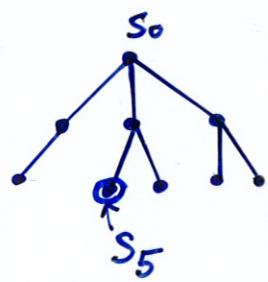
current state	tape read	⇒	operation	next state
start → S_0	b		R	S_0
S_0	1		R	S_1
S_1	1		R	S_1
S_1	b		Write 1	S_2
S_2	1		L	S_3
S_3	1		L	S_3
S_3	b		R	S_4
S_4	1		write b	S_5 ← stop

b 1 1 1 b 1 1 b ⇒ b b 1 1 1 1 1 b

(3) (2) (5)

- **Deterministic** vs. **Nondeterministic** Turing Machine

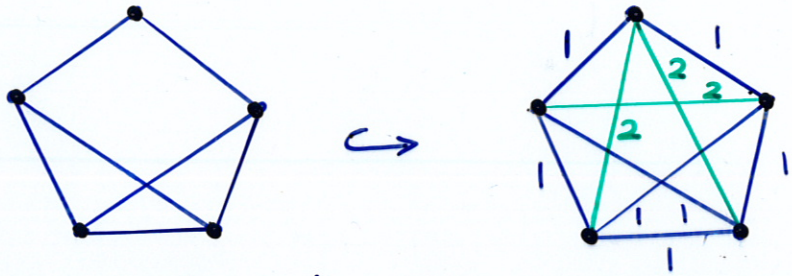
$(S, b) \rightarrow \{ (R, S_0), (L, S_1), (1, S_j), \cdots \}$



- Computation time : # steps used
  Memory : # tape cells
- Polynomial vs. Exponential time
  (tractable) (intractable)
- NP problem : polynomial time computable by NDTM
  P problem : 〃 DTM

- $A \hookrightarrow B$, problem A is **polynomial reducible to** problem B.
  - Travelling Salesman (T): Find a tour of all cities of distance $\leq M$
  - Hamilton Cycle (H): Find a simple cycle includes all vertices

$H \hookrightarrow T$



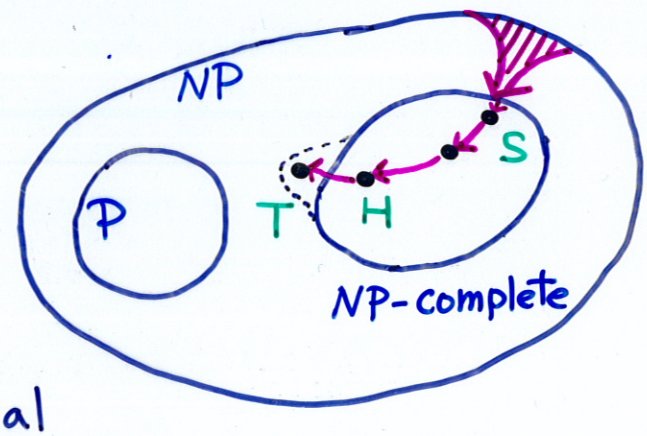
$\exists$ a simple cycle $\Leftrightarrow$ A tour $\leq N$ (# cities)

- Have a polynomial time algorithm for T
  $\Rightarrow$ " H (H: NP-complete $\Rightarrow$ T also)
- problem A is **NP-complete** (Hardest) if $\begin{cases} A \in NP \\ \forall\ NP \text{ problems} \hookrightarrow A \end{cases}$

  i.e., Have polynomial algorithm for A $\Rightarrow$ all NP problems are tractable
- S: Satisfiability problem:

  $\exists$ assignment of 0, 1 to x_i, s.t., a given formula is 1 ?

  $(x_1+x_3+x_5)*(x_1+\bar{x}_2+x_4)*(\bar{x}_3+x_4)$



Cook: S is NP-complete

Karp: A large class of combinatorial problems are NP-complete

Turing Award !

- Given problem Q: can't find polynomial time algorithm,

  then $R \hookrightarrow Q$ $\Rightarrow$ Q is NP-complete
  (R: NP-complete) i.e., Q as hard as many famous problems

P = NP ?

NP-complete : Example 2.

- Given problem (A) is NP-complete.

  (A): Given integers $c_1, c_2, \ldots, c_N$ and K,

  Is there subset $S \subset \{1, 2, \ldots, N\}$, $\rightarrow \sum_{j \in S} c_j = K$?

- Prove that problem (C) is NP-complete also.

  (C): Given integers $d_1, d_2, \cdots, d_N$

  Is there subset $S \subset \{1, 2, \ldots, N\}$, $\rightarrow \sum_{j \in S} d_j = \sum_{j \notin S} d_j$?

Proof: (1) $(C) \in NP$

(2) $(A) \hookrightarrow (C)$

- Given $c_1, c_2, \ldots, c_N, K$, $\quad (M = c_1 + \cdots + c_N)$
- Let $d_1 = c_1$, $d_2 = c_2$, $\ldots$, $d_N = c_N$,

  $d_{N+1} = 2M$, $\quad d_{N+2} = 3M - 2K$
- Then

$$\exists S, \sum_{j \in S} c_j = K \iff \exists S' \subset \{1, 2, \ldots, N+2\}$$

$$\sum_{j \in S'} d_j = \sum_{j \notin S'} d_j$$

pf: "$\Rightarrow$" $S' = S \cup \{N+2\}$

"$\Leftarrow$" $N+1$ or $N+2 \notin S'$

if $N+2$ then $S = S' - \{N+2\}$

$$\underbrace{c_1 + c_2 + \cdots}_{K} + \underbrace{\cdots + c_N}_{M-K} = M$$

K : $3M - 2K$

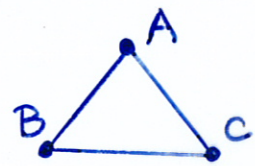
$M-K$: $2M$

Exhaustive Search

- Intractable, what to do? (Example: Travelling Salesman: Shortest simple cycle thru all nodes)
  - Special cases
  - Exhaustive search : Backtracking + Recursion
    - Branch and Bound (Heuristics)
    - Dynamic Programming
  - Approximation algorithm : Nearly-optimal solution
  - Probability analysis
    - Hard instances are rare
    - Probabilistic (randomized) algorithm:  in program
      - Las Vegas algorithm
        correct, $1-\varepsilon$ polynomial time
        ε exponential time
      - Monte-Carlo algorithm
        polynomial time, $1-\varepsilon$ correct
        ε incorrect

- **Backtracking**
  - Depth-First search + recover environment before exit.

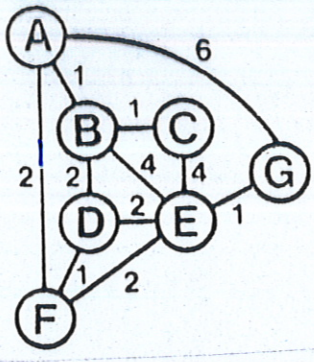
```
Visit (K)
    id := id+1;  val[k] := id;
    for each son t of K do
        if val[t] = 0 then visit(t);
    id := id-1;  val[k] := 0;
```



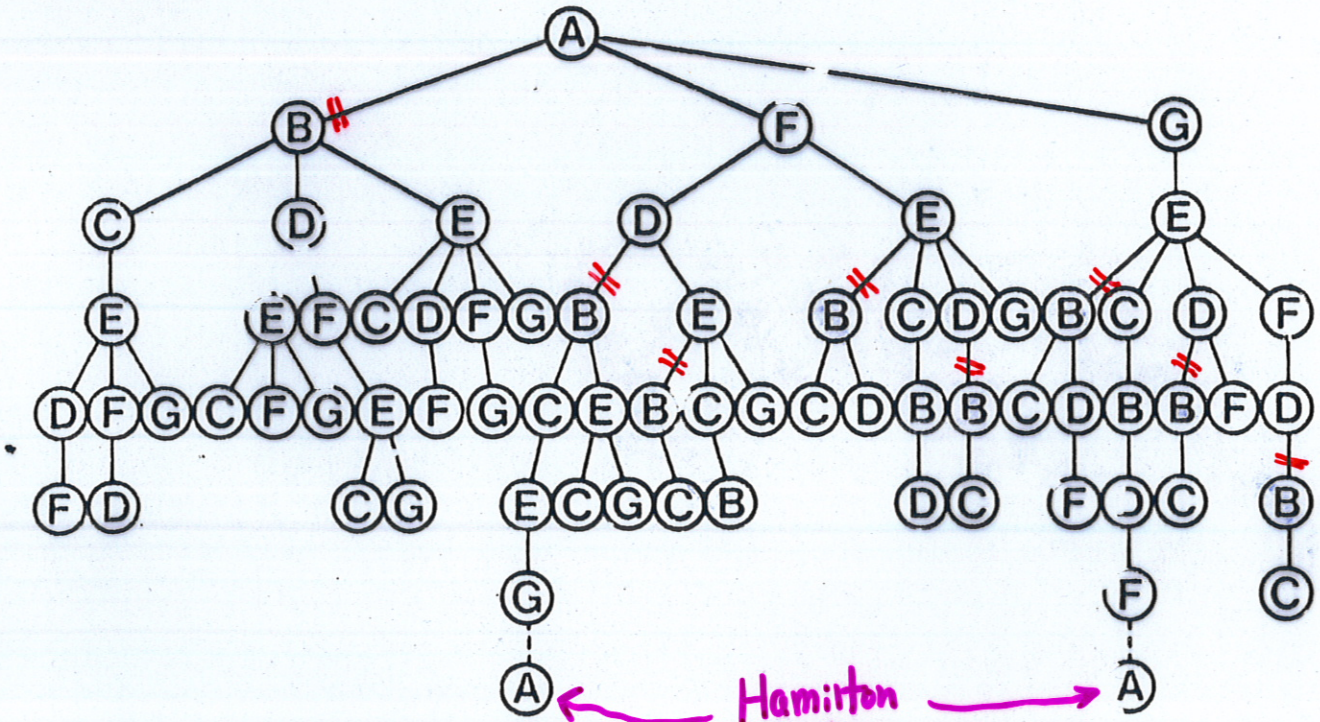
A-B-C
A-C-B

- **Hamilton cycle**
  - Have simple cycle connects all vertices?

$\Leftrightarrow \exists K$, $val[k] = V$ and
$K \to 1$

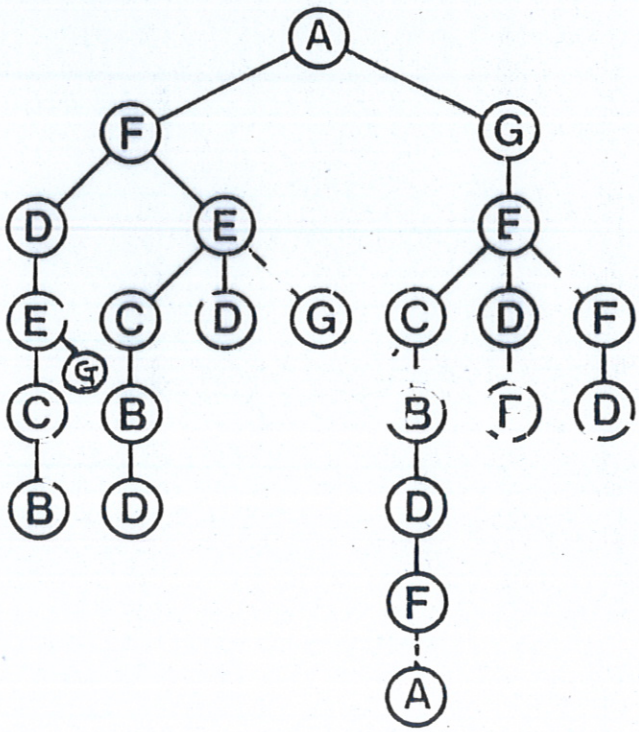


search tree



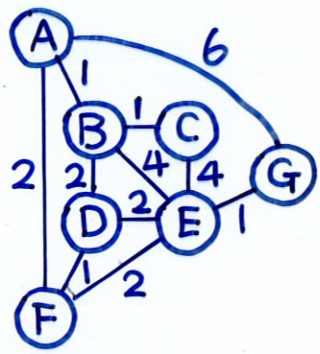
Prunning search tree: to limit search using Heuristics.

e.g.: stick to order A → C → B → A



Travelling dog:

- Find shortest simple path connects all nodes



- Exhaustive search + Prunning techniques:

(1) Branch & Bound:

A F D B C E G = 11

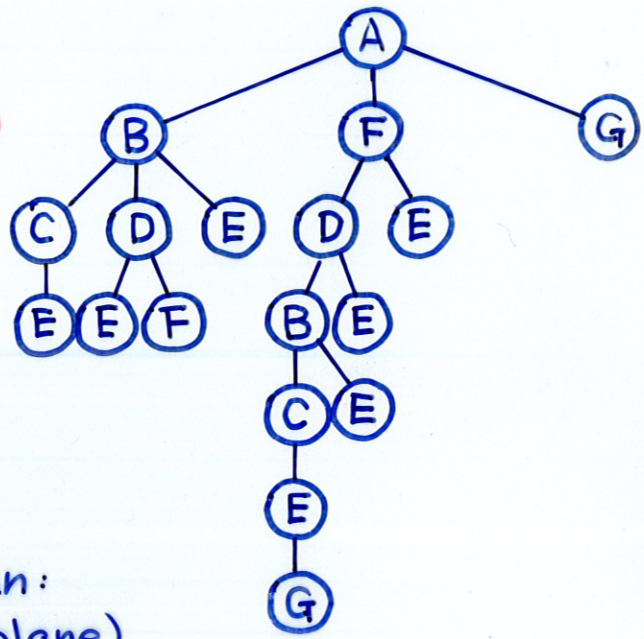
A G E B = 11 already STOP

(2) ∀ edge ≥ 1, 7 nodes totally

A G = 6 STOP

(3) Partial solution + MST ≥ 11 STOP

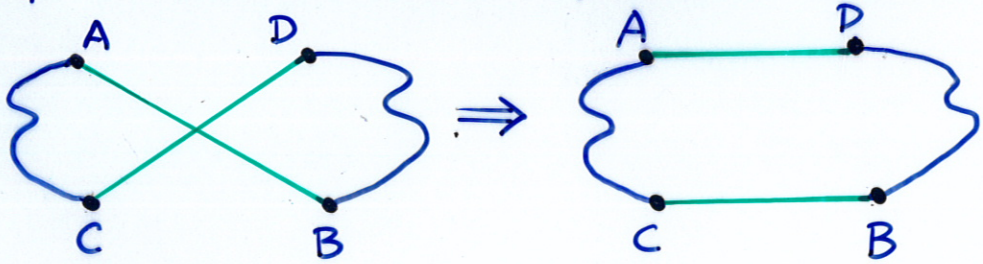
(4) Partial path + the rest nodes not connected (A B E) STOP



(5) Others:

For Euclidean travelling salesman:

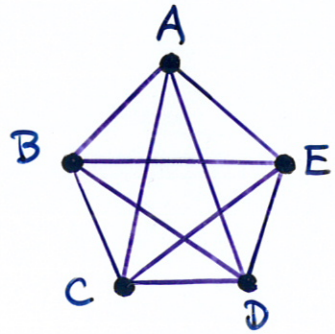
(graph distance = distance in plane)



- **Generate permutation**

generate N! permutations by exhaustively searching N-node complete graph

```
for K := 1 to N do val[k] := 0;
id := -1; Visit(0);
Visit(K)
  id := id+1; val[k] := id;
  if id = V then Writeperm;
  for t := 1 to V do
      if val[t] = 0 then visit(t);
  id := id-1; val[k] := 0;
```



Probabilistic Algorithm

- Problem: Is n prime ?
  - No polynomial algorithm.
  - Can't prove it doesn't exist !

- Theorem. (Wilson) p: prime $\Leftrightarrow (p-1)! \equiv -1 \pmod p$

- Theorem. (Euler) p: prime, $1 \le a \le p-1$ $\Rightarrow a^{p-1} \equiv 1 \pmod p$

- Proposition. n: composite $\Rightarrow \left| \{ a \mid a^{n-1} \equiv 1 \pmod n,\ 1 \le a \le n-1 \} \right| < \frac{n-1}{2}$

$$a^{\frac{n-1}{2}} \equiv \left(\frac{a}{n}\right) \bmod n$$

- Algorithm

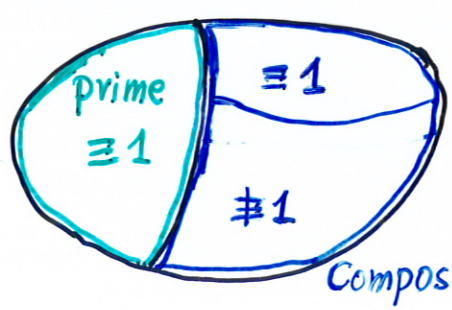
```
TestPrime(n)
    Choose a ∈ {1, 2, ..., n-1} randomly;
    if (a, n) > 1 then return(composite);
    if a^(n-1) ≠ 1 (mod n) then return(composite)
                           else return(prime);
```

  - efficient

- Input n $\Rightarrow$ output: composite (correct)
  : prime (correct : $Pr \ge \frac{1}{2}$)
  wrong : $Pr < \frac{1}{2}$

- Repeat 100 times

$$Pr(\text{wrong}) < \frac{1}{2^{100}} \quad !!!$$



- **Branch and Bound**
  - Branch – most promising path (based on bound)
  - Bound – bound of possible solutions > best solution so far
    ⇒ pruned

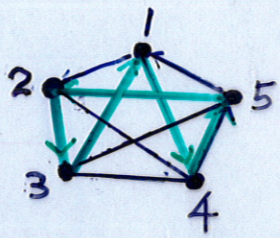
Example

$$\begin{bmatrix} \infty & 14 & 4 & 10 & 20 \\ 14 & \infty & 7 & 8 & 7 \\ 4 & 5 & \infty & 7 & 16 \\ 11 & 7 & 9 & \infty & 2 \\ 18 & 7 & 17 & 4 & \infty \end{bmatrix}$$

- **Lemma**

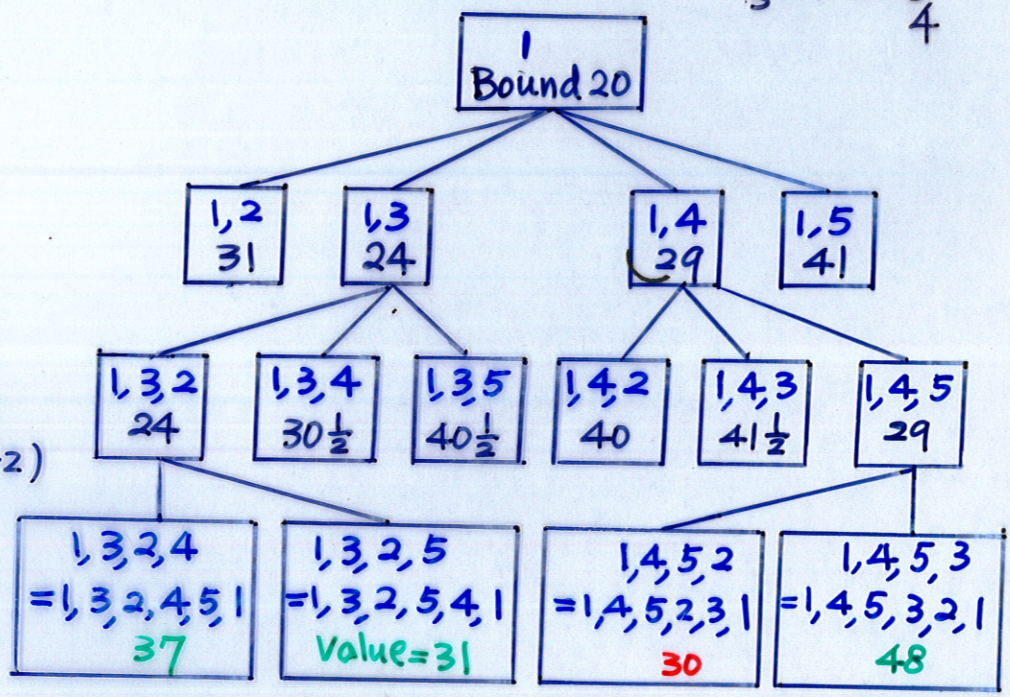
$$2 \cdot \text{shortest path} \geq \sum_{1 \le K \le N} \left\{ \min_i (C_{iK}) + \min_j (C_{Kj}) \right\}$$

$$\Rightarrow \text{shortest path} \geq \sum_{1 \le K \le N} \left\{ \min_i \frac{C_{iK}}{2} + \min_j \frac{C_{Kj}}{2} \right\}$$



$1 \to \cdots \to 1$ Bound $= \frac{1}{2}(4+7+4+2+4+4+5+4+4+2) = 20$

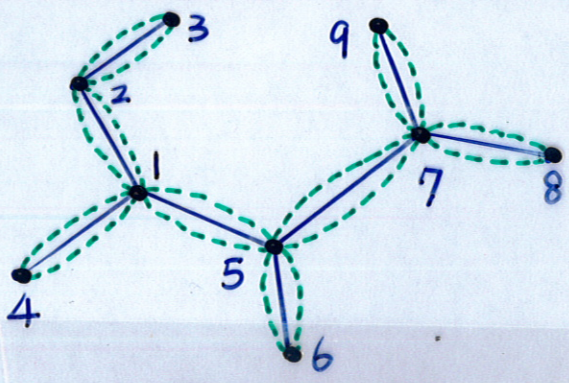
$$\begin{bmatrix} \infty & 14 & \infty & \infty & \infty \\ 14 & \infty & 7 & 8 & 7 \\ 4 & \infty & \infty & 7 & 16 \\ 11 & \infty & 9 & \infty & 2 \\ 18 & \infty & 17 & 4 & \infty \end{bmatrix}$$



1-2, Bound $= 14 + \frac{1}{2}(7+4+2+4+4+7+4+2) = 31$

$$\begin{bmatrix} \infty & \infty & 4 & \infty & \infty \\ 14 & \infty & \infty & 8 & 7 \\ \infty & 5 & \infty & \infty & \infty \\ 11 & \infty & \infty & \infty & 2 \\ 18 & \infty & \infty & 4 & \infty \end{bmatrix}$$

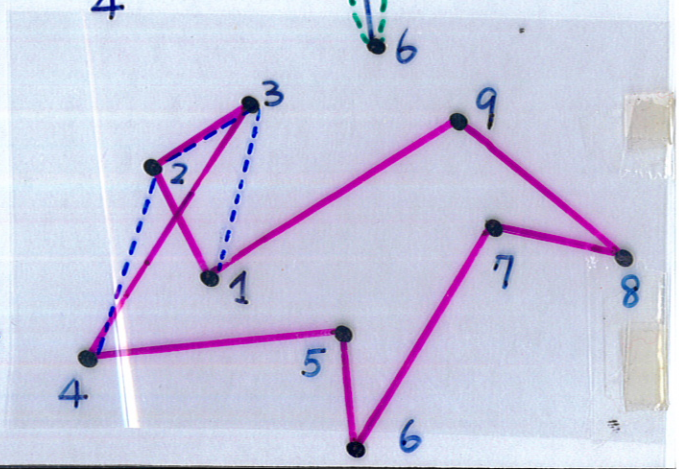
1-3-2, Bound $= 4+5+\frac{1}{2}(7+2+4+11+4+2) = 24$



- **Approximation algorithm** (planar pts)
  (1) Construct Minimum Spanning Tree
  (2) Shortcut to get a tour

  ⇒ MST < TSP ≤ approximate tour ≤ 2MST < 2TSP

  (3) Can even improve!



Travelling Salesman Problem

Find shortest simple cycle connects all nodes. ($N!$ cycles)

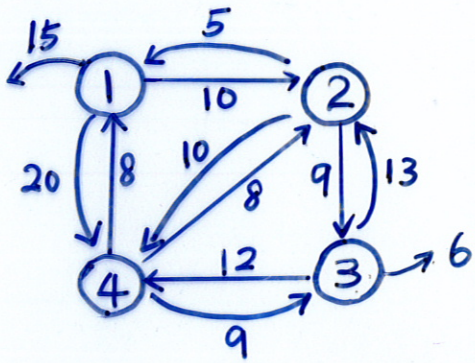
- **Dynamic Programming**

$g(i, S)$:= Shortest path $i \overset{S}{\rightsquigarrow} 1$ (thru nodes in S only)

- $TSP = g(1, V-\{1\}) = \min_{2 \le j \le N} \{ c_{1j} + g(j, V-\{1, j\}) \}$

$$g(i, S) = \min_{j \in S} \{ c_{ij} + g(j, S-\{j\}) \}$$

$$g(i, \phi) = c_{i1}, \quad i = 2, 3, \cdots, N$$



Example:

(1) $g(2, \phi) = 5$
$g(3, \phi) = 6$
$g(4, \phi) = 8$

(2) $g(2, \{3\}) = c_{23} + g(3, \phi) = 9 + 6 = 15$
$g(2, \{4\}) = c_{24} + g(4, \phi) = 10 + 8 = 18$
$g(3, \{2\}) = 13 + 5 = 18$
$g(3, \{4\}) = 12 + 8 = 20$
$g(4, \{2\}) = 8 + 5 = 13$
$g(4, \{3\}) = 9 + 6 = 15$

$$[c_{ij}] = \begin{bmatrix} 0 & 10 & 15 & 20 \\ 5 & 0 & 9 & 10 \\ 6 & 13 & 0 & 12 \\ 8 & 8 & 9 & 0 \end{bmatrix}$$

(3) $g(2, \{3,4\}) = \min\{ c_{23}^{=9} + g(3, \{4\})^{=20}, \; c_{24}^{=10} + g(4, \{3\})^{=15} \} = 25$
$g(3, \{2,4\}) = \min\{ c_{32}^{13} + g(2, \{4\})^{18}, \; c_{34}^{12} + g(4, \{2\})^{13} \} = 25$
$g(4, \{2,3\}) = \min\{ c_{42}^{8} + g(2, \{3\})^{15}, \; c_{43}^{9} + g(3, \{2\})^{18} \} = 23$

(4) $g(1, \{2,3,4\}) = \min\{ c_{12} + g(2, \{3,4\}), \; c_{13} + g(3, \{2,4\}), \; c_{14} + g(4, \{2,3\}) \}$
$= \min\{ 10+25, \; 15+25, \; 20+23 \} = 35$

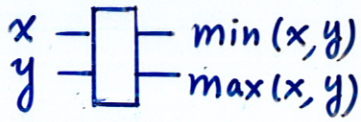
$\Rightarrow TSP = c_{12} + c_{24} + c_{43} + c_{31}$
$= 10 + 10 + 9 + 6 = 35$

$$T = (N-1) + (N-1) + \sum_{K=1}^{N-2} (N-1) \binom{N-2}{K} K$$

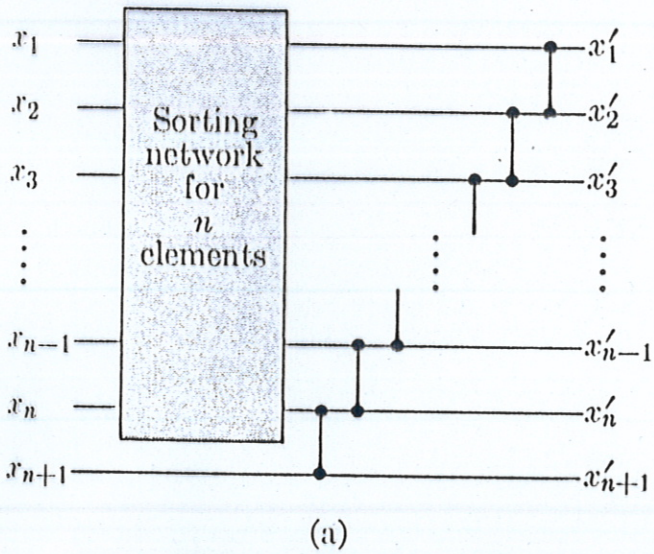
$$= 2(N-1) + (N-1) \sum_{K=1}^{N-2} K \binom{N-2}{K} = \Theta(N^2 2^N)$$

Parallel Algorithms

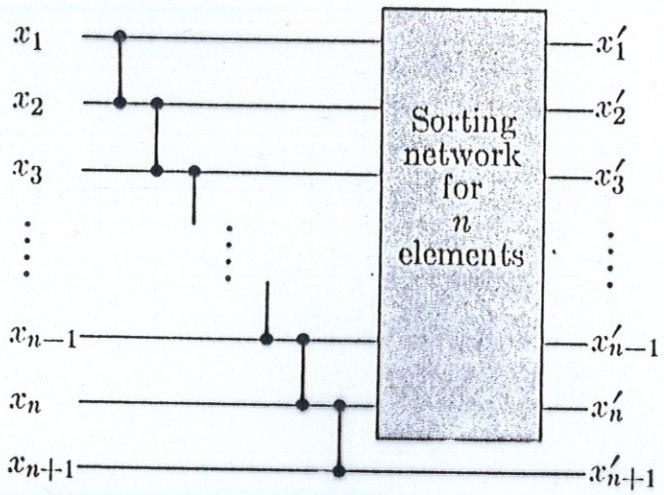
Comparator



Algorithm Machine : Hardware implementation of Algorithm



(a)

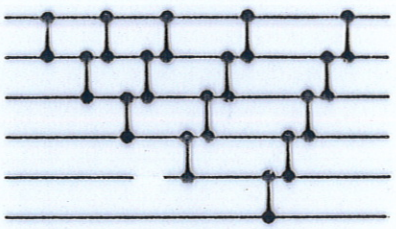


(b)

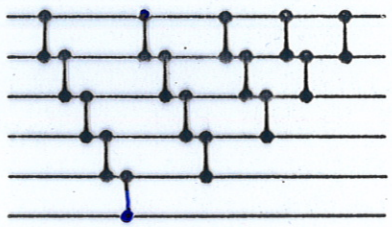
Insertion Sort

Bubble Sort

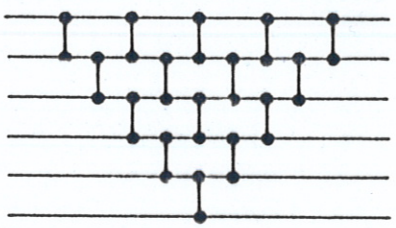
N=5



(a)



(b)



With Parallelism Insertion Sort = Bubble Sort

Odd-Even Merge and Perfect Shuffle

Theorem $\begin{cases} a_1 \le a_2 \le a_3 \le \cdots \le a_{2N} \\ b_1 \le b_2 \le b_3 \le \cdots \le b_{2N} \end{cases}$, 2 sorted sequences to be merged

Odd Merge $\{a_1, a_3, \cdots, a_{2N-1}\}, \{b_1, b_3, \cdots, b_{2N-1}\} \to \{c_1, c_2, \cdots, c_{2N}\}$ recursively

Even Merge $\{a_2, a_4, \cdots, a_{2N}\}, \{b_2, b_4, \cdots, b_{2N}\} \to \{d_1, d_2, \cdots, d_{2N}\}$

$c_1, \underline{d_1 : c_2}, \underline{d_2 : c_3}, \cdots, c_{2N-2}, \underline{d_{2N-2} : c_{2N-1}}, \underline{d_{2N-1} : c_{2N}}, d_{2N}$: sorted

proof: $\begin{matrix} c_{i+1} \\ d_i \end{matrix}$ are $\begin{matrix} 2i \\ 2i+1 \end{matrix}$ smallest.

1 3 (4) (10) 11 14 15 16
2 5 6 7 8 9 12 13

5th odd c_5 — 4th even d_4

Example:

A E G G I M N R
A B E E L M P X

$\Rightarrow$ A B E E I M N R
A E G G L M P X

$\Rightarrow$ A B E E
I M N R
A E G G
L M P X

$\Rightarrow$ A B E E
A E G G
I M N R
L M P X

$\Rightarrow$ A B
E E
A E
G G
I M
N R
L M
P X

$\Rightarrow$ A B
A E
E E
G G
I M
L M
N R
P X

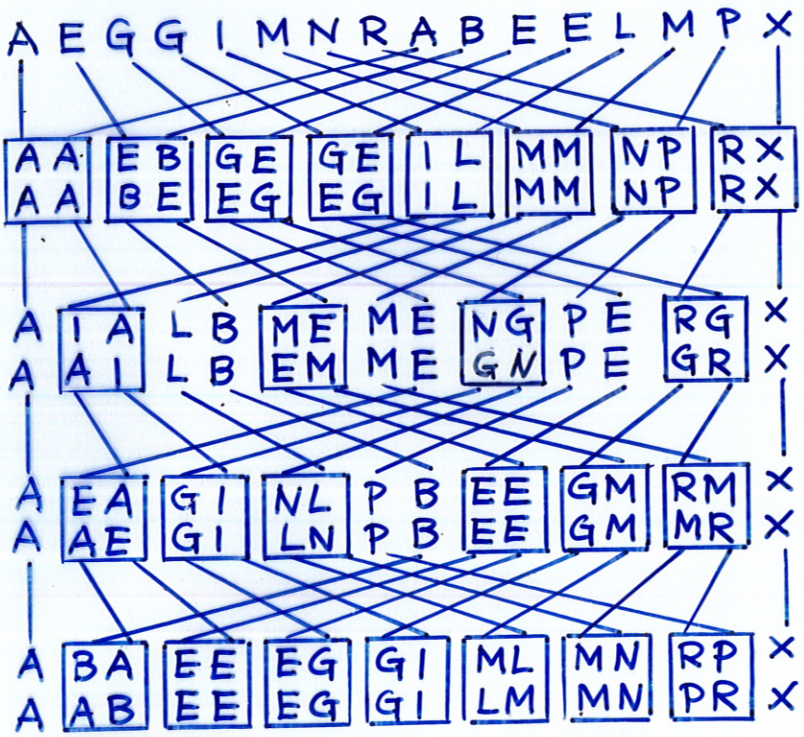
perfect shuffle

A
B
A
E
E
G
G
I
M
L
M
N
R
P
X

$\Rightarrow$ A A B E E E G G I L M M N P R X

Even merge

Odd merge



$\log N$ parallel steps

$N \log N$ comparators

Bitonic Merge

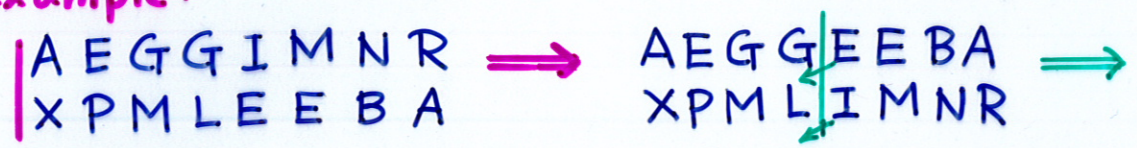
- **Bitonic sequence** $\begin{cases} (1)\ a_1 \le a_2 \le \cdots \le a_i \ge a_{i+1} \ge \cdots \ge a_{2N} \quad \text{OR} \\ \quad\ \ge\ \ge\ \ge\ \le\ \le\ \le \\ (2)\ \text{can cyclically shift to (1)} \end{cases}$

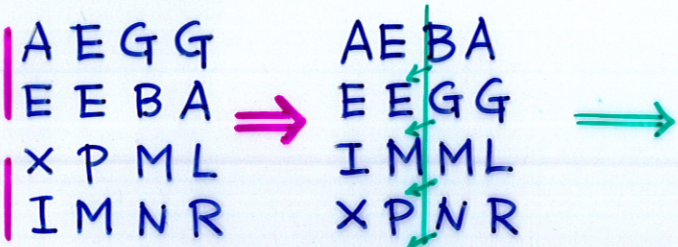
- **Theorem** $\begin{bmatrix} a_1 \\ a_{N+1} \end{bmatrix} \begin{bmatrix} a_2 \\ a_{N+2} \end{bmatrix} \cdots \begin{bmatrix} a_N \\ a_{2N} \end{bmatrix} \rightarrow \begin{matrix} \{c_1, \cdots, c_N\} \\ \{d_1, \cdots, d_N\} \end{matrix}$ i.e., $c_i = \min\{a_i, a_{i+N}\}$, $d_i = \max\{a_i, a_{i+N}\}$

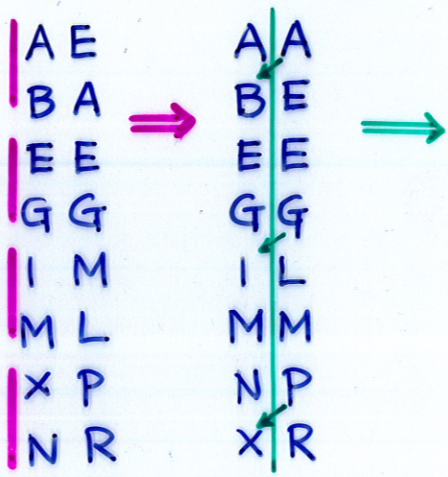
$\Rightarrow$ (1) $c_i \le d_j \quad \forall i, j$

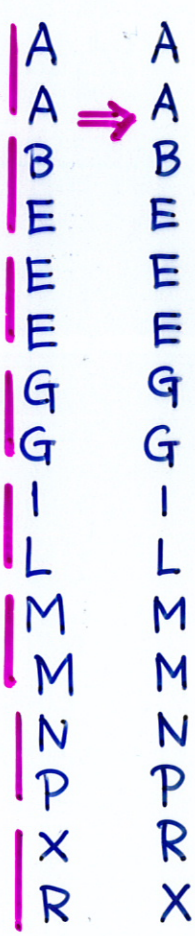
(2) $\{c_i\}, \{d_i\}$ are both bitonic. (recursively merge)

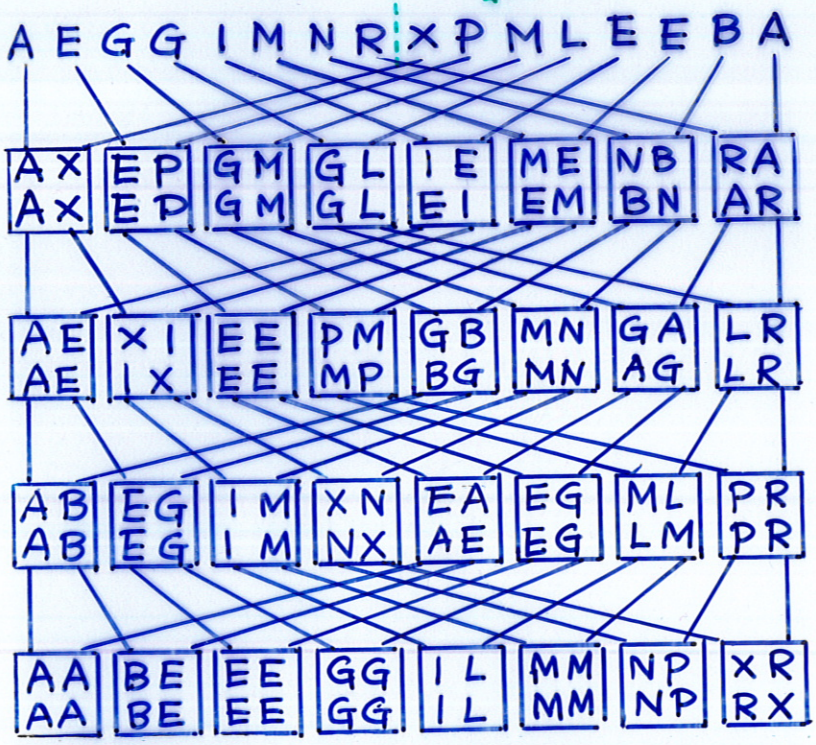
- **Example:**

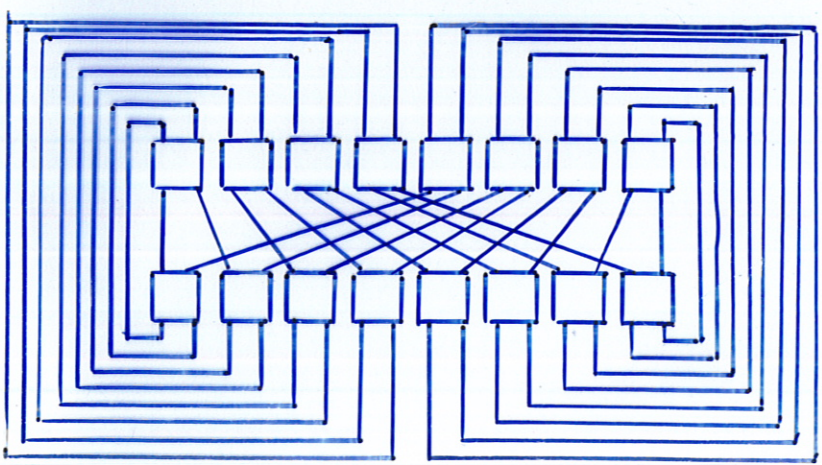










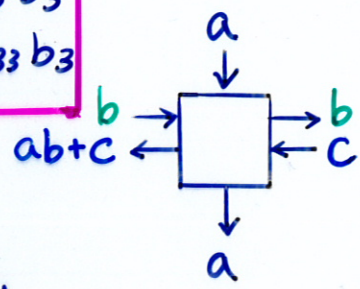


Systolic Array (H.T. Kung, 孔祥重)

- Simple, identical cells mesh-connected in a regular way
- communicate with adjacent cells only.

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix}$$

$$c_1 = a_{11} b_1 + a_{12} b_2 + a_{13} b_3$$
$$c_2 = a_{21} b_1 + a_{22} b_2 + a_{23} b_3$$
$$c_3 = a_{31} b_1 + a_{32} b_2 + a_{33} b_3$$



a_{33}

a_{23} × a_{32}

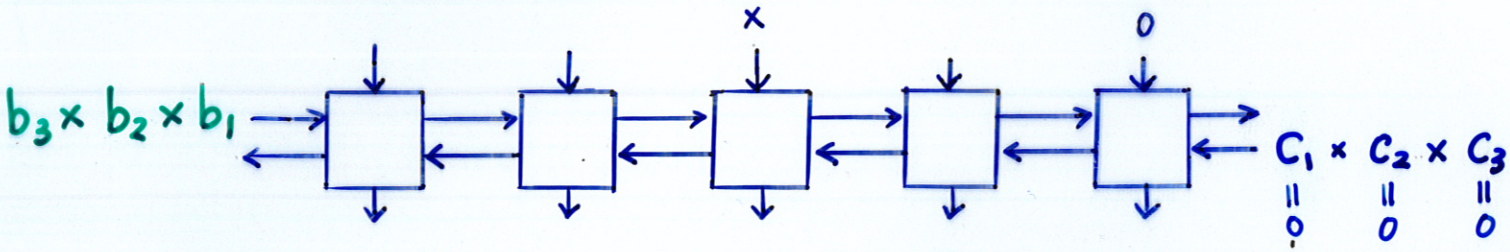
a_{13} × a_{22} × a_{31}

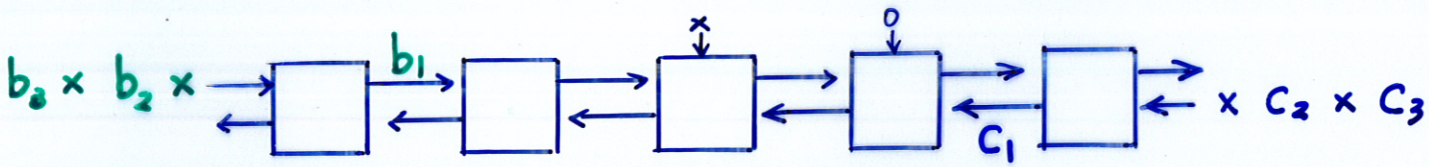
a_{12} × a_{21}

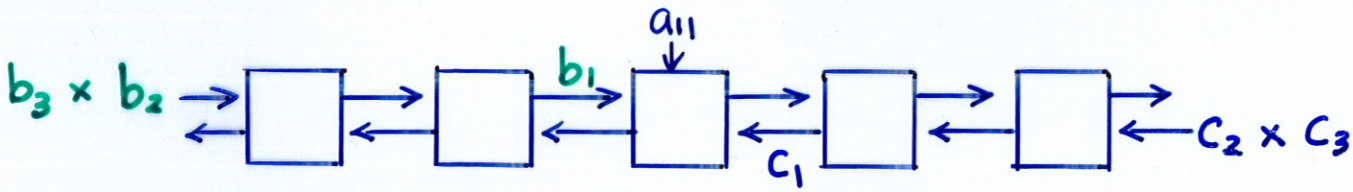
a_{11}

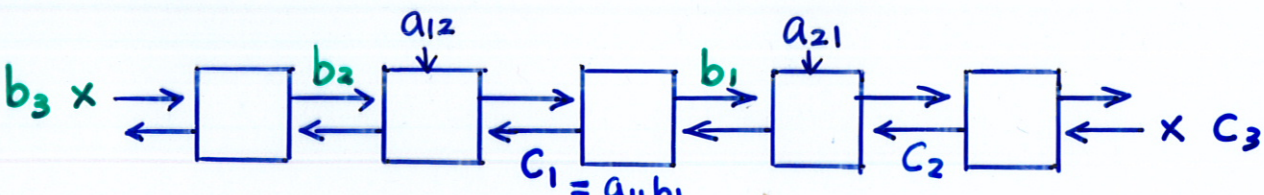
×

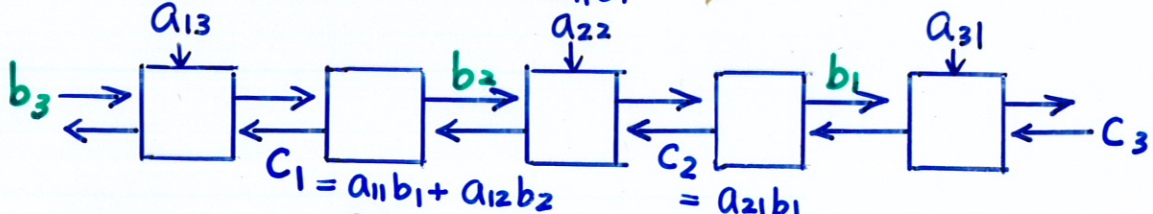
×

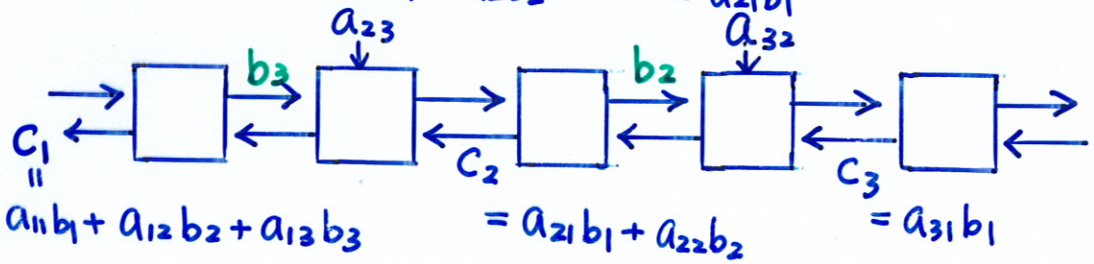












$2N-1$ cells

$4N-2$ steps

Fast Fourier Transform (FFT)

Compute Discrete Fourier Transform (DFT) in $O(N \log N)$ time

- **Theorem** $p(x) = a_0 + \cdots + a_{N-1}x^{N-1}$, $q(x) = b_0 + \cdots + b_{N-1}x^{N-1}$, $\exists\, x_1, \cdots, x_N \rightarrow$ $p(x_1) = q(x_1), \ \vdots, \ p(x_N) = q(x_N) \Rightarrow p(x) \equiv q(x)$ i.e., $a_i = b_i$

- Given $x_1, x_2, \cdots, x_N$, 2 representations for $p(x) = a_0 + a_1 x + \cdots + a_{N-1}x^{N-1}$
  - (i) $[a_0, a_1, \cdots, a_{N-1}]$
  - (ii) $[p(x_1), p(x_2), \cdots, p(x_N)]$?

- Choose $1, \omega, \omega^2, \cdots, \omega^{N-1}$, where ω: principal N^{th} root of unity

$$\begin{cases} \omega^N = 1 \\ 1 + x + x^2 + \cdots + x^{N-1} = 0, \ \forall x = \omega, \omega^2, \cdots, \omega^{N-1} \end{cases}$$

Example: (1) $\omega = e^{\frac{2\pi i}{N}}$

(2) $\omega = 2$, $N = 3$

$\because \ 2^3 \equiv 1 \bmod 7$

$1 + 2 + 2^2 \equiv 0 \bmod 7$

- $\vec{a} = [a_0, a_1, \cdots, a_{N-1}]$ or $p(x) = a_0 + \cdots + a_{N-1}x^{N-1}$

DFT of $\vec{a}$ $= DFT(\vec{a}) = [p(1), p(\omega), p(\omega^2), \cdots, p(\omega^{N-1})]$

$$= \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{N-1} \\ 1 & \omega^2 & \omega^4 & \cdots & \omega^{2(N-1)} \\ \vdots & & & & \\ 1 & \omega^{N-1} & \omega^{2(N-1)} & \cdots & (\omega^{N-1})^{N-1} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{N-1} \end{bmatrix}$$

$= W \cdot \vec{a}$ $\qquad W[i, j] = \omega^{ij}, \quad 0 \le i, j < N$

- **Theorem**

$DFT^{-1} = W^{-1} = \frac{1}{N}(\omega^{-ij})_{0 \le i, j < N}$, another DFT !

- **Applications**
  - DFT in signal processing
  - Polynomial Multiplication
  - Integer Multiplication : $p(x) \cdot q(x)$, then $x = 2$

• Polynomial Multiplication

$p(x) = a_0 + a_1 x + \cdots + a_{N-1} x^{N-1}$, $\quad q(x) = b_0 + b_1 x + \cdots + b_{N-1} x^{N-1}$

$p(x) \leftrightarrow \vec{a} = [a_0, a_1, \cdots, a_{N-1}, 0, \cdots, 0] \leftrightarrow [p(1), p(w), \cdots, p(w^{N-1}), \cdots, p(w^{2N-1})]$

$q(x) \leftrightarrow \vec{b} = [b_0, b_1, \cdots, b_{N-1}, 0, \cdots, 0] \leftrightarrow [q(1), q(w), \cdots, q(w^{N-1}), \cdots, q(w^{2N-1})]$

$p(x) \cdot q(x) \leftrightarrow \vec{a} \otimes \vec{b} = [c_0, c_1, \cdots, c_{2N-1}] \leftrightarrow [p(1)q(1), p(w)q(w), \cdots, p(w^{2N-1})q(w^{2N-1})]$

Convolution $\quad c_k = \sum_{0 \le i \le k} a_i b_{k-i}$ $\quad$ w : $2N^{th}$ root of unity

• FFT

$$p(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5 + a_6 x^6 + a_7 x^7$$
$$= (a_0 + a_2 x^2 + a_4 x^4 + a_6 x^6) + x (a_1 + a_3 x^2 + a_5 x^4 + a_7 x^6)$$
$$= p_e(x^2) + x\, p_o(x^2)$$

$w = w_8 = e^{\frac{2\pi}{8} i}$ $\quad 8^{th}$ root of unity

$w^4 = -1$

$p(w^0) = \quad = p_e(w^0) + w^0 p_o(w^0)$
$p(w^4) = p_e(w^8) + w^4 p_o(w^8) = p_e(w^0) + w^4 p_o(w^0)$

$p(w^1) = \quad = p_e(w^2) + w^1 p_o(w^2)$
$p(w^5) = p_e(w^{10}) + w^5 p_o(w^{10}) = p_e(w^2) + w^5 p_o(w^2)$

$p(w^2) = \quad = p_e(w^4) + w^2 p_o(w^4)$
$p(w^6) = p_e(w^{12}) + w^6 p_o(w^{12}) = p_e(w^4) + w^6 p_o(w^4)$

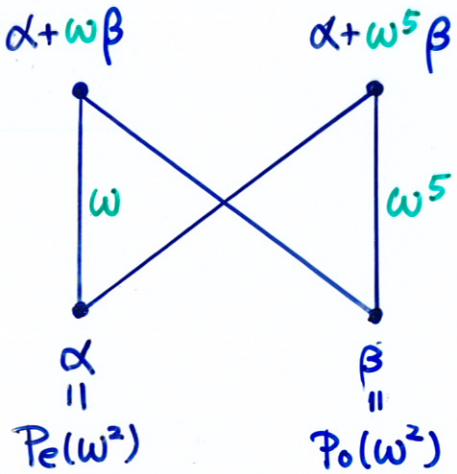
$p(w^3) = \quad = p_e(w^6) + w^3 p_o(w^6)$
$p(w^7) = p_e(w^{14}) + w^7 p_o(w^{14}) = p_e(w^6) + w^7 p_o(w^6)$
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad = -w^3$

- Evaluate $p(w_8^i)$, $0 \le i < 8$ $\quad$ (DFT, length = 8)

↪ Evaluate $\begin{cases} p_e(w_8^{2i}) = p_e(w_4^i) \\ p_o(w_8^{2i}) = p_o(w_4^i) \end{cases}$ $\quad 0 \le i < 4$ $\quad$ (" , length = 4)

recursively

$w_4 = w_8^2$ = 4^{th} root of unity



Butterfly

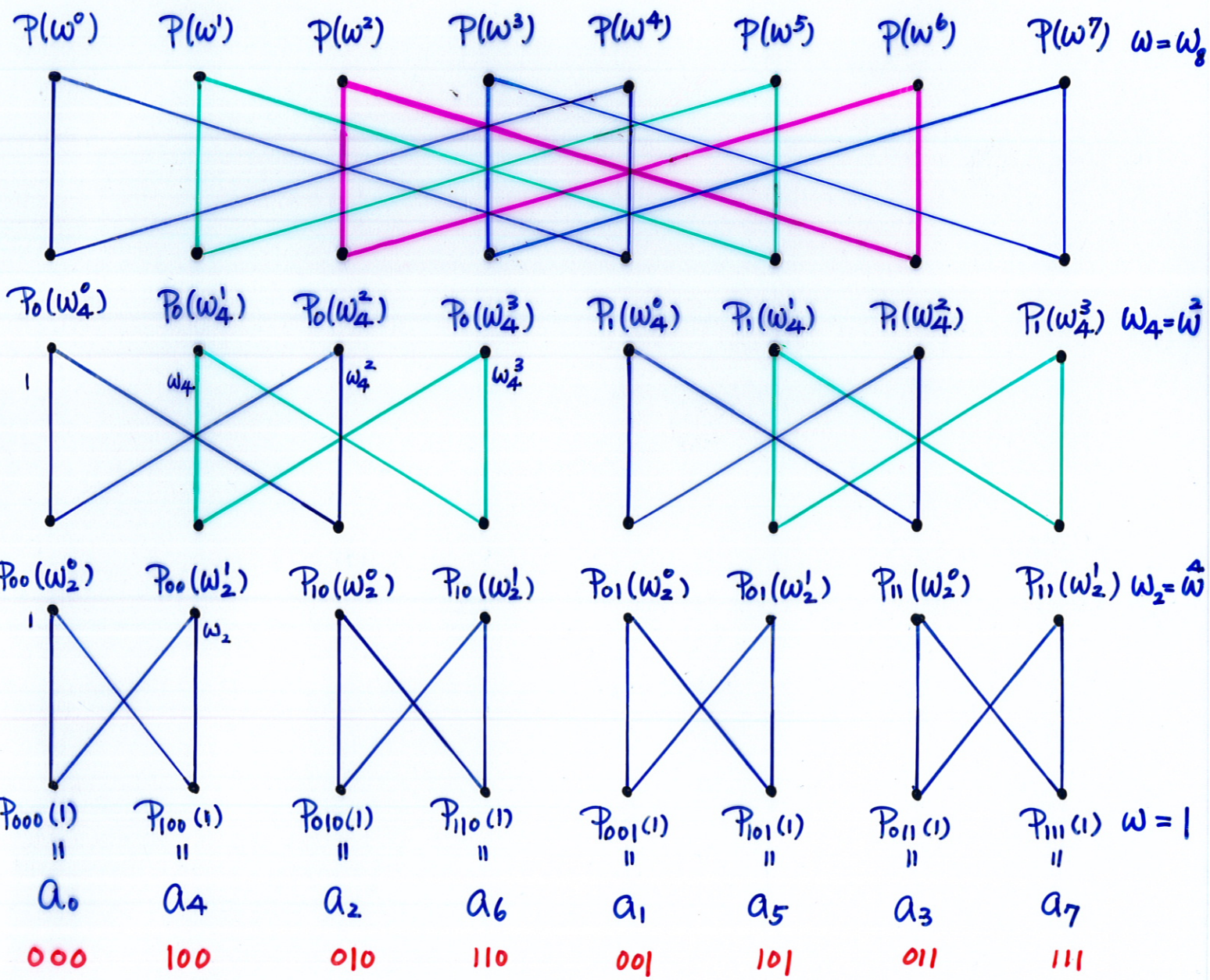
- Apply FFT recursively to $P_e(x) = a_0 + a_2 x + a_4 x^2 + a_6 x^3$

  $P_o(x) = a_1 + a_3 x + a_5 x^2 + a_7 x^3$

0	1	2	3	4	5	6	7
0	2	4	6	1	3	5	7
0	4	2	6	1	5	3	7
0	4	2	6	1	5	3	7

$P_o(x) = P_e(x)$: Even function

$P_1(x) = P_o(x)$: Odd function

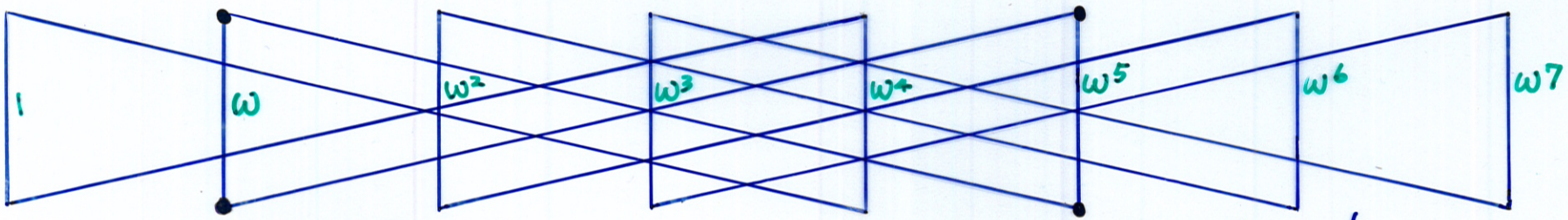


- Why DFT^{-1} is another DFT?
- $T = O(N \log N)$ sequentially, OR $T = \log N$ parallel steps, $N \log N$ processors

$P(\omega^1) =$
$(a_0 + a_2\omega^2 + a_4\omega^4 + a_6\omega^6)$
$+ \omega(a_1 + a_3\omega^2 + a_5\omega^4 + a_7\omega^6)$

$P(\omega^5) =$
$(a_0 + a_2\omega^2 + a_4\omega^4 + a_6\omega^6)$
$+ \omega^5(a_1 + a_3\omega^2 + a_5\omega^4 + a_7\omega^6)$

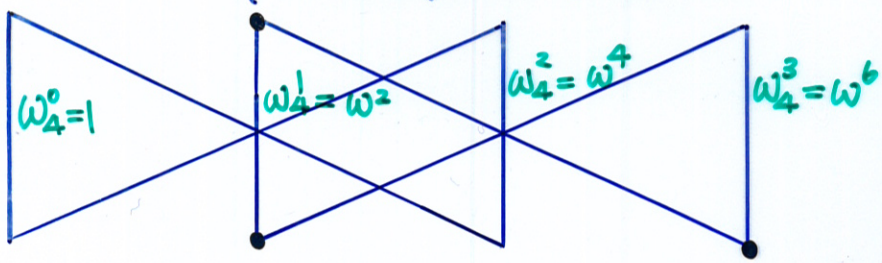
$\omega = \omega_8$

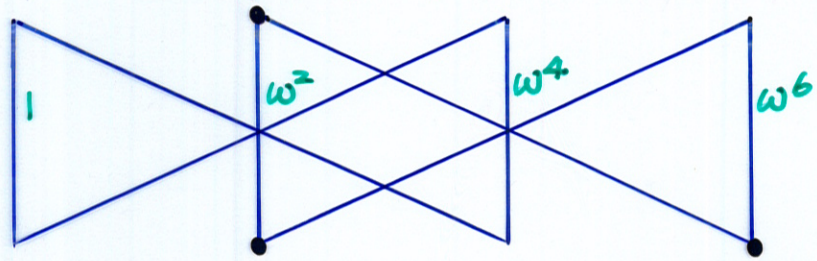


$a_0 + a_2\omega^2 + a_4\omega^4 + a_6\omega^6$
$= (a_0 + a_4\omega^4) + \omega^2(a_2 + a_6\omega^4)$

$a_1 + a_3\omega^2 + a_5\omega^4 + a_7\omega^6$
$= (a_1 + a_5\omega^4) + \omega^2(a_3 + a_7\omega^4)$

$\omega_4 = \omega^2$

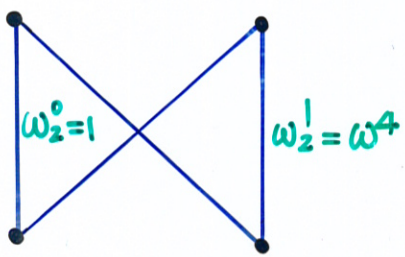


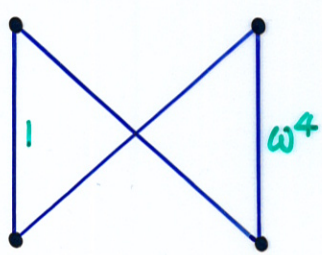


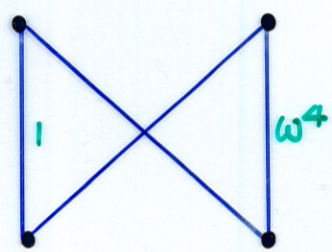
$a_0 + a_4$ $\quad a_0 + \omega^4 a_4$ $\quad a_2 + a_6$ $\quad a_2 + \omega^4 a_6$

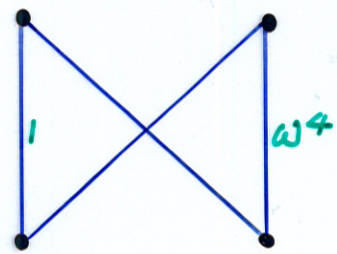
$a_1 + a_5$ $\quad a_1 + \omega^4 a_5$ $\quad a_3 + a_7$ $\quad a_3 + \omega^4 a_7$

$\omega_2 = \omega^4$









a_0 $\quad a_4$ $\quad a_2$ $\quad a_6$ $\quad a_1$ $\quad a_5$ $\quad a_3$ $\quad a_7$